PC-8801/PC-9801

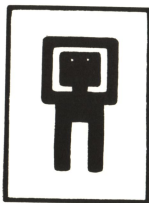
BASIC㊙ハイテク100選

涌井良幸
涌井貞美 ★著

誠文堂新光社

PC-8801/PC-9801

BASIC㊙ハイテク100選

涌井良幸
涌井貞美★著

誠文堂新光社

序

現代社会において，パソコンは社会のすみずみまで普及しその影響力を次第に大きくしている．10年ほど前パソコンが初めてアメリカから輸入され始めた頃，それはごく少数の研究者やマニアのための道具であった．時がたつにつれパソコンは，パーソナルなコンピュータとして個人の限定された利用形態を脱皮し，事務処理，学習，研究，ゲーム等に広い応用性を見い出されたのである．

このように広く用いられるようになったパソコンに対して大切なものはソフトウェアである．それは汎用コンピュータとまったく同じである．どんなに立派なコンピュータでもソフトウェアなくしては何の役にも立たない．逆にいうと優れたソフトウェアは社会の貴重な財産となる．そのソフトウェアを作成する言語としてパソコンには多くの場合 BASIC が用いられる．BASIC はほんの少しの英語と数学の教養をもっているものには大変覚えやすいというメリットがある．そしてそれはハードウェアやシステム全般についての知識をほとんど必要としなくても使うことができるのである．しかし，まさにこの BASIC の特徴が色々な問題を生んでいるのである．すなわち，あまりコンピュータ自身の勉強をしなくてもシステムは稼動する．またプログラミング技法を学習せずとも一応プログラムは動くのである．そのため，実際にパソコン内で稼動しているプログラムを見ると，あまりにコンピュータの動作原理やプログラミング技法を無視しているものが多く発見される．処理効率を考えず，使いやすさを配慮せず，ただ力ずくでコーディングしたものがしばしば見つけられるのである．

既述のようにパソコンは個人の利用のためというにとどまらず深く社会に浸透しており，そのソフトウェアは社会の財産になるものである．それを作成者個人しか理解できないように作ったのでは，パソコンの社会的要請に応えられないであろう．効率を無視していたり，修正要求に迅速に応えられないようで

は困りものである．本書は，このような社会のパソコン・ソフトへの要請に十分対応できるような知識を提供するためのものである．すなわち，有名なプログラミング技法やアルゴリズム，コーディングのしかた等を誰にでも理解できるように解説したものなのである．

本書は読みやすいように各節は各々独立している．また節の順番に従って読む必要もない．気の向いたところ，必要なところから読んで頂きたい．また本書に記載されているプログラム例は NEC PC-8801/9801 のための BASIC を基本としている．当然，本書の内容はマシンに依存するものではない．他機種については その各々のマニュアルを参照して随時修正して頂きたい．

本書の理解には入門程度の BASIC の知識を仮定している．そのため細かい命令の解説は省いてある．BASIC を初めて学ぶ方が本書を利用されるときは，必ず BASIC の入門書を座右に置いてもらいたい．

本書が少しでもパソコンのプログラミング技法の向上に役立てられれば幸いである．しかし，本書を読んだからといってすぐに良いプログラムが作れるわけではない．大切なのは優れたプログラムを作ろうとする意志であり，そのための学習意欲である．十分その辺のことを確認しておいて頂きたい．

著　者

目　　次

第 5 章 グラフィック命令活用法

第1章

優れたプログラムの設計のために

我々がプログラムを作ろうと思うとき，まずその設計をするわけであるが，ここでは，そのための基礎知識を提供する．すなわち，プログラムはどのように作られるべきか，そのためにはどんな手段と知識とが必要なのか，といったことを概説する．

1 優れたプログラムとは

我々がプログラムを作るとき当然優れたものを作ることをねらうであろう．しかし，プログラムに関して“優れている”ということを真に定義するのは大変である．何をもって優れていると評価するかは，場合によって違うからである．ここでは“優れている”ということの一般的な目安を記述してみよう．この目安を確認した上で，ケースバイケースに優れたプログラムを作成することを追求してもらいたい．

優れたプログラムの条件として，まず掲げられるのが

(1) 正確である

ということ．どんな立派なプログラムでもこれなくしては意味をなさない．たとえば金銭計算で1円の計算ミスも許されないのである．しかし正確だからといってもそれだけですむものではない．当然，次のことも必要である．

(2) 使いやすい

プログラムを使うのに，いちいちマニュアルを見ながら使うプログラムでは利用者は大変困る．また，一つの処理をするのに何時間も待たされたのではたまったものではない．たとえばオセロ・ゲームをするとき，人間が考える以上にパソコンが考え込んでしまっては，パソコン・ゲームは楽しみではなくイライラさせるだけのものになってしまう．処理速度は十分大きくなくてはならない．

以上の2点は誰もが納得するところである．何年か前までのパソコンのプログラムは，実際にこれ以上のものは要求されなかった．しかしパソコンが社会に深く浸透し，それに対する要求も多様になってきた現在，この(1)，(2)をみたすだけでは決して良いプログラムとは呼べないようになっている．我々は次の要求をさらに求める．

(3) 分かりやすいこと

(4) 間違いにくいこと

(5) 追加・訂正がしやすいこと

これら三つは，互いに関連し結び合わさっている概念である．分かりやすいこ

とは間違いにくいことであり，また間違いにくく良く整理されているために追加・訂正がしやすいのである．

これら三つの要求が，なぜ現在のパソコン・プログラムに求められているのかを説明しよう．まず第一に，バグの発見のしやすさが掲げられる．パソコンのプログラムといっても，以前のような単純な論理のものだけを扱っていればよいわけではない．社会の要請に従って，複雑な処理をせねばならなくなってきている．この複雑な処理に対して，当然プログラムは長くなり，むずかしくなってきている．人間の思考が不可避的に誤りを伴う以上，その所産であるプログラムにも誤りがある．その誤りを長く複雑なプログラムの中に発見し，すぐに対処できるようにするのは容易ではない．それを少しでも容易にしようというのが(3)，(4)，(5)の努力である．

第二の理由として，パソコンのプログラムはその作った本人だけが理解していればよい，という時代ではなくなってきたことである．パソコンは既にパーソナルなものではなくなってしまったので，一人の人間が作成したプログラムは，他の多くの人に見られ利用され，拡充されるようになっている．それはまたパソコンが社会の財産になる条件でもある．このとき，自分にだけ分かる“汚ない”プログラムを書いていては，それを利用したり修正して拡充してゆこうという他の人の要求を疎外することになってしまう．現代においては，誰が見ても分かるプログラムを作ることは非常に大切であり，それが(3)，(4)，(5)の方針となって現われているのである．

その他にも色々な理由が掲げられるが，ここでは，これだけを強調すれば十分であろう．我々がプログラミングするとき，常にここで述べた(1)〜(5)のことを意識していなければならないであろう．

優れたプログラムを作成しようと常に努力してプログラミングしよう．

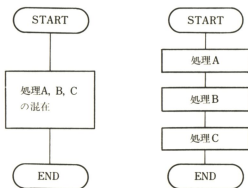
2 プログラムはモジュール化しよう

優れたプログラムの要件として

(1) 分かりやすい

(2) 修正しやすい

ということがある．この二つを実現する代表的な手法として，プログラムの**モジュール化**というものがある．モジュールとは本来基本的な部品とか構成要素のことをいうが，我々がプログラムを作るとき部品から製品を作り上げるように，いくつかの論理的なまとまりから，プログラム全体を構築せよというのが「プログラムのモジュール化」でいわんとすることである．

この要請を図示すると次のようになる．

この二つのフローチャートで右側の構造をプログラムに与えよ，というのである．左のような構造にすると次のような欠陥をもってしまう．

(1) デバッグがしにくい

(2) 機能追加・修正がしにくい

モジュール化することで一つひとつのモジュールに責任をもたせることができ，また部品を入れ換えるように機能修正ができるのである．プログラムが大きく

なればなるほど，このプログラムのモジュール化という思想は，我々に合理的なプログラム設計の指針を与えてくれるものである．

プログラムをどのようにモジュール化すべきかは，対応する問題に依存するが，その原則となるものは，分割したモジュール相互の関係ができるだけ独立するようにすることである．もしそれらが独立しておらず，互いに呼び合うことが多いと，かえってプログラムは難解になってしまう．

プログラムをモジュール化して設計すると，また次のようなメリットが生まれる．長いプログラムを一人で作成するのは大変だが，それを複数の人間が共同して作ろうと思うとなおさら大変なことが多い．一般的にパソコンのプログラマーは，共同作業に不慣れなため，いざ共同でプログラミングしようとすると，非常に繁雑になる．それを回避する手段の一つが，プログラムのモジュール化である．プログラムをいくつかの独立した処理に分割しておけば，やりとりするデータ構成をしっかり打ち合わせておくことで，容易に共同作業が行えることになる．　•

プログラムはダラダラと長くせずに，論理的に独立したモジュールで組み立てよう．

Memo　**ミクロなモジュール化**

我々がプログラムをコーディングしてゆくとき，上下の命令および隣り合う命令は互いに強い論理的な関係をもっているべきである．ただ無批判に命令を並べてゆくことは厳に慎しむべきである．そうすると，でき上がったプログラムの一つのモジュール内においても，色々な独立なまとまりができることになる．まさにミクロなモジュール化がなされるわけである．我々は，このようにプログラムを組み立てることで，修正要求に容易に対応でき，またバグ発生に対しても迅速な対処ができるのである．

3 プログラムには振り出し点を作ろう

大きなプログラムを作るときまず考えるのがプログラムのモジュール化（2節参照）である．一つのプログラムを論理的に別々なパートに分け，それを組み合わせて全体を作るのである．その組み合わせのしかたとして，次のような方法を覚えておくことは有益である．すなわち，**すべてのモジュールはプログラムの一点にもどるようにせよ**という設計方法である．プログラムを構成するモジュールは，論理的に独立しているから，それらは対等であるが，その各モジュールを管理する**メインルーチン**を作るのである．各モジュールは，そのメインルーチンから呼び出され，処理が終われば再びそのメインルーチンにもどる，という方法をとるのである．そして，そのメインルーチンに帰るとき，そのもどり点となるところを各モジュールに共通させておくようにするのである．

このような構造をとることで次のようなメリットが生じる．

(1) メインルーチンが資源を管理することでシステム全体の管理が楽に

なる．

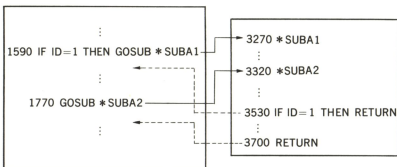
(2) 割り込み (STOP，HELP，FUNCTION KEY の割り込み) に対して対処しやすい．

(3) バグ発生に対して対応がとりやすい．

この方法は双六(すごろく)に振り出し点があるのに似ている．振り出し点があることで全体は見やすくなり，それがあることで困ったときにそこにもどれるのである．

きちんとしたメイン・ルーチンをつくると，プログラム全体が見やすくなる．

Memo **モジュールは1入口，1出口**

メインルーチンから各モジュールを呼ぶときの原則，および復帰するときの原則は1入口，1出口である．すなわち次のような呼び方，帰り方をしてはならないということである．

この図を見れば分かるように，複数の出入口が一つのモジュール（いまの場合サブルーチン）にあると非常に見にくくなるのである．

4 BASICの特徴をつかもう

優れたプログラムを作成するためには，まずそのプログラムを記述する言語の特徴を知っておかねばならない．BASIC の特徴を知るには次のプログラムを考えるのが早い．

```
100 S=0:I=1
110 IF I>100 THEN 150
120 S=S+I
130 I=I+1
140 GOTO 110
150 PRINT "S=";S
160 END
```

これは1から100までの整数の和を求めるプログラムである．これをBASIC がどのように処理してゆくかを考えてみる．RUN コマンドが実行されると行番号100を BASIC は見る．そして何が書かれているかを調べ（解読），それに従って実行する．次に行番号110が見られる．ここでも BASIC による解読と，それに続く実行が組み合わされる．行番号140までこのような処理のもとに一直線にたどりつくが，次に行番号140が解読・実行されると，再び行番号110へもどる．そして行番号110から140までが100回解読・実行をくり返されて，このループが終了する．注意せねばならないことは「解読」という操作が常に伴われているということである．

コンピュータは，2進数しか理解できない．そこで，BASIC で記述された命令は，2進数に解読され，そこで始めて実行されるのである．そして BASIC は，すべての BASIC 命令を1ステップずつ解読してゆく．したがって，ループ計算では同じ命令の解読を何回でもやることになる．これは，命令の解読（翻訳）を一気に行い，後はその結果を用いて実行するコンパイル言語と大きく違う点である．BASIC プログラムの実行は，英語の不得手な生徒（CPU）が辞書（BASIC）を片手に必死に英文を和訳しているようなものなのである．この特徴をしっかりつかんだ上で，プログラムの設計およびコーディングをせねばならない．

BASIC は翻訳に手間どる言語である，ということを忘れないように．

Memo **コンパイル言語とインタプリタ言語**

コンピュータは2進数しか解読できず，したがって，BASIC のように人間の言葉に近い言語で書かれたプログラムは，その2進数に何らかの形で翻訳されなければならない．(コンピュータが理解する2進数の並びで記述されたプログラムを**機械語プログラム**と呼んでいる．それに対してBASIC などは人間に近い言語でプログラムを記述するため，そのような言語を**高級言語**と呼んでいる．)

我々は，翻訳のしかたから高級言語を二つに分類している．一つは**コンパイル言語**であり，一つは**インタプリタ言語**である．コンパイル言語にはFORTRAN, COBOL 等汎用コンピュータで使用されているものの多くが含まれる．この言語は，まず記述されたプログラムを一気に翻訳し，コンピュータの理解できる2進数の並びからなる機械語プログラムを作成してしまう．そして実際に RUN させるのは，この機械語プログラムで行うのである．それに対してインタプリタ言語は，1命令1命令を一つひとつ翻訳してゆく．まさに同時通訳的な言語である．したがって，翻訳時間が計算処理時間の大半をしめてしまうのである．

パソコンに効率の悪いインタプリタ言語であるBASIC が採用されているのは二つの理由による．一つはコンパイル言語のように新たな機械語プログラムを作成するものでは，その新たなプログラムを収納する領域がパソコンにはない，ということである．もう一つの理由はデバッグが容易である，ということである．

5 バグ対策は綿密に

人間には誤りがつきものである．特にコンピュータのように複雑なシステムに致ってはそれが不可避的ともいえる，何千，何万行にわたるプログラムにおいては，誤りがまったくないと思う方が不自然であろう．

我々がプログラムを作成しようとするとき，始めから完璧なものを作ろうと思うと肩がこる．上述のように複雑なものには，誤りが伴うということを始めから受け入れてしまえばよい．そして次のような発想をとることを勧める．

どうせ誤りを犯すなら，その誤りがすぐに発見できるように設計しよう．

パソコンにおいて，誤りと呼ばれるものに次の五つがあげられるだろう．

(1) ハードエラー

(2) モニタエラー

(3) 文法エラー

(4) 論理エラー

(5) 入出力エラー

ハードエラーとは,ハードウェアすなわちパソコンの機械そのものの誤り(故障）である．モニタエラーとはメーカの作成したプログラム（モニタまたはシステムプログラムなどと呼ばれる）の中の誤りである．文法エラーとは，我々がBASIC言語を用いたときに犯した文法上の誤りである．論理エラーとは，プログラム作成者が設計およびコーディング時に犯した論理上の誤りである．入出力エラーとは，オペレータがキー操作を誤ったとか，パソコンと他のコンピュータとをオンラインで結合したとき，その回線上で発生した雑音，などのことである．

(2), (3), (4)の誤りのことを，しばしばバグ(bug，虫)という．そしてその誤りを発見し，訂正することをデバッグ(debug，虫とり)という．我々がプログラムを設計し，コーディングするとき，一番神経を使わねばならないのは，このバグ対策である．

ハードエラーやモニタエラーについては，パソコンの通常な使用法においては対策を考える必要はない．また入出力エラーについては，その入出力エラーにどんな種類があり，プログラムはその各々に対して，どのような対応をとるべきかを考えておかねばならないが，その対応さえしっかりしていれば，入出力エラーはエラーであることをやめる．また(3)の文法エラーについては，プログラムを実行すると BASIC がそれを見つけてくれるので容易に対処できる．

これらのエラーに対して，我々パソコンのプログラムを作成する人間が，対応に苦慮するのは(4)の論理エラーである．正しいと思って設計し，またそれをコーディングしたものの誤りであるから，容易にその発見がなされない．そのための対応として2節のプログラムのモジュール化があり，また第8章のデバッグ対策があるわけであるが，本節では次のことだけを強調しておこう．

完璧なものをねらわず，バグの発見しやすい論理を用いよう．

Memo デュアルシステム

我々のプログラムにエラーがあることを前提とするなら，メーカが作成したモニタ（システム・プログラム）にも誤りがあるはずである．またハードウェアにもミスがあることも考えられる．我々が誤りを前提としたシステムを考える以上，当然，我々の作成したプログラムとともにシステム固有のモニタにも誤りがあることを仮定した対応を考えておかねばならない．金融機関や原子炉などで用いられているコンピュータシステムは，高い信頼性が要求されているために，通常このハードエラーやモニタエラーをも考えに入れて，複数のコンピュータを同時に動かしている．こうすることで一方が障害を起こしても他方が動けるようにするのである．このように並列することで，信頼性を向上しようとしたシステムをデュアルシステムという．

6 プログラムの大きさを見積ろう

設計したプログラムを紙上にコーディングし，それを見ながらプログラムをキーインしている途上，もしくは全部キーインし終わって RUN させたとき，

memory overflow

というメッセージが出力されたという話はよく聞かれる．せっかく設計しコーディングしたプログラムを再び設計変更し，コーディングし直すというのは大変面倒な作業である．したがって，設計段階でそのプログラムの大きさを予想しておくことは大切である．もし予想したプログラムの大きさが，使っているパソコンのメモリ数より大きければ，その段階で設計変更が可能である．

プログラムの大きさを，設計段階で見積るのは困難で，多分に経験がものをいう．すなわち，いま作成しているプログラムが，どれくらいのステップ数(行番号の数）になり，どれくらいの変数領域が必要となるかは，正確には通常把握できるものではない．ここでは，もしそのステップ数や変数の種類がおおざっぱにつかめたとしたとき，このプログラムがどれくらいのバイト数を必要とするかの計算方法を示そう．

まずプログラム領域について考える．予想したステップ数をＮ，1 ステップ（1 行番号）の平均の命令文字長を l とすると，

$$プログラム領域 \fallingdotseq N \times (l+5) \quad バイト$$

括弧内の＋5 については，あくまで概数である．この数値を 1,000で割った数値がキロバイト（KB）数となる．

次に数値変数について考えてみよう．整数形変数名の数を Nz 個，平均の変数名の文字数を lz 個とし，また単精度および倍精度の変数名の数，その文字数を各々 Ns，ls，Nd，ld とすると，

$$数値変数領域 \fallingdotseq Nz \times (lz+3) + Ns \times (ls+5) + Nd \times (ld+9) \quad バイト$$

文字変数については，文字変数名の個数を Nc 個，その変数名の長さの平均を lc，また実際にその変数に納められる文字データの平均の長さを lr とすると

$$文字型変数領域 \fallingdotseq Nc \times (lc + lr + 5) \quad バイト$$

以上のようにしておおまかにプログラムの大きさを予想することができる．次に現システムにおいて，我々ユーザがどれくらいのメモリを使うことができるのかを調べる方法を示そう．マニュアルに 64KB とか 128KB とかのメモリの大きさの数値が記されているが，実際にはそのマニュアルの数値よりもかなり小さいメモリしか，我々は使用できない．特にディスクを多用するシステムはそのことがいえる．したがって，現在どれくらいのメモリがあまっているのかを知ることは大切である．

我々が使えるメモリの大きさを知る手段として，BASIC は

```
FRE (k)
```

という関数を用意してくれている．ここで k=0 では未使用変数領域のバイト数を関数値としてとり，k=1 では未使用プログラム領域のバイト数を関数値としてとる（注．この k の値に対する機能については機種により多少異なる）．

もし見積ったプログラムの大きさが，この関数で調べた値よりも大きければ当然設計変更をしなくてはいけない．この設計変更の際，なるべくならば既に作成した設計書を大幅に変えたくはない．そこで次のような変更をまず考え，なるべくもとの案を生かすべきだろう．

(1) CHAIN 命令を用いてプログラムを分割する．

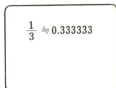
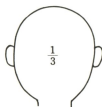
(2) 使用ずみ資源はすぐに返却（ERASE，CLOSE 命令など）する．

(3) 変数を整理し，また整数型ですむ変数はその宣言をしておく．

(4) データ形式を圧縮し，ビット演算をさせることで論理を変えずにデータ領域を節約する．

プログラムの大きさを見積ってコーディングしよう．それをサボるとかえって時間を損することがある．

7 データの性質を把握しよう

コンピュータは，我々人間の頭のように無限桁の数値を扱うことは不可能である．扱うデータは，すべて有限桁の数値として表現し，その数値で計算を実行する．たとえば有名な例として，我々は1を3等分した数1/3を考えることができるが，コンピュータは，それを近似としてしか認識できない．

このようにコンピュータに特有な限界をしっかり認めておかないと，我々の作るプログラムは無意味なものになってしまう．

次のプログラムを考えよう．

```
100 N=50
110 FOR K=1  TO N
120    CLS
130    PRINT "社員番号=";K
140    INPUT "総支払額=";S
150    INPUT "税　　金=";TAX
160    INPUT "社会保険=";PENS
170    PAY=S-TAX-PENS
180    LPRINT "no=";K;"        支給額=";PAY
190 NEXT K
200 END
```

これは給料の総支払い額から税金，社会保険料を引いた額を計算し出力するという原始的なプログラムである．現在の通常のサラリーマンに対してはこのプログラムで正常に動く．しかし，もしこの社員の中に高給取りがいて，7桁を超えた額が支払われたとすると，プログラムは誤った結果を出力する．たとえば社員番号20番の人の給料の総支払額が2,000,000円，税金が500,000円，社会

保険料が199,999円だったとすると，当然その手当額（支給額）は1,300,001円となるはずである．これを実際に計算させると次のようになる．

```
NO=20  支給額=1.3E+06
```

すなわち，2百万円という額はパソコンは桁数が大きくて覚えきれなかったのである．

上例は非常に幼稚な例であるが，このような誤りは日常しばしば起こることである．すなわち設計の段階でしっかりと入力データの見積りをしないために起こる誤りなのである．上例がもししっかりした見通しのもとに設計され，コーディングされたなら次の宣言文が第1行にあるべきである．

```
DEFDBL  A-Z
```

こうすることで，上の誤りは回避できる．すなわち変数を倍精度にするのである．

しかし，常に倍精度計算で解決できるとは限らない，倍精度にするとデータの納まるメモリ領域が2倍となり，また計算スピードも遅くなってしまう．できることなら単精度の方がよいのである．

このように，ソフトウェアの作成において決まった手法があるわけではなく，場合に応じて我々は対応してゆかねばならない．そして，その各々について，データの性格をしっかりつかみ，適格なデータ形式を選んでいかねばならないのである．

設計段階で入力データの上限，下限，精度をしっかり調べよう．

8 データの訂正がしやすい設計を

我々がコンピュータを使うとき，キー操作のミスは日常茶飯事である．したがって，入力ミスのたびに“しまった”と思わせるプログラムでは，我々は大変疲れてしまうであろう．誤っても「大丈夫，すぐに直せる！」と思わせるものでなければならないのである．すなわち，データ訂正のしやすいプログラムを作成せねばならないのである．

データ訂正のしかたとして次の三段階がある．

(1) キー操作中にミスに気づいたときに訂正する．

(2) キー入力がすべて終了したときに訂正する．

(3) プログラム実行中または実行後に訂正する．

この三つの段階の各々について，訂正をサポートできるようにしなくてはいけない．

(1)の段階では，まず入力データのチェックをプログラムが厳しく行うことが必要である．そうすれば，プログラムの責任でエラーの回復ができる．また次の入出力例に示されたような方法も有名である．

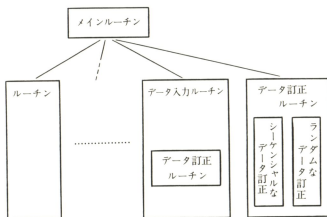
```
NO  804464        氏名  山田太郎
入社年月日        50年4月1日
年齢              30
職種コード        A4

                              OK(y/n)?
```

これはある会社の社員のデータとする．一人の区切りがついたところでOKかどうかのチェックをするのである．こうすることで，こまめにデータ訂正ができ，利用者に安心感を与えられるのである．

(1)の段階でいくらチェックしても必ずエラーが入り込む．そのための対応として(2)の段階がある．このための準備として全入力データの一覧表が要求される．すなわち，打ち込んだデータがまとめて出力されたものである（44節参照）．その一覧表と元本とをつき合わせて(2)の段階のチェックが行われ，そして訂正箇所が判明する．そしてその判明した訂正箇所について，一つひとつかつ迅速に訂正できるルーチンを組むべきである．

(1)，(2)の段階を経ても，どうしてもエラーが潜み込むものである．それはプログラム実行中ないしは実行後に判明する．我々はこのような突発的なデータ入力エラーの発見に対しても，すぐに対応できるようなエラー処理を考えておかねばならない．

(2)，(3)についてはルーチンを共用できるが，経験上(2)の訂正はデータ入力順に行えることが便利であり（シーケンシャルな訂正），(3)についてはまったくランダムな訂正ができると便利である．これを図示したのが下図である．

9 マニュアルを完備しよう

パソコンとは，本来個人を対象にした計算機であったが，現代においては広く色々に人々に使用される．したがって，ひとたびあるプログラムが人々の利用に供されたなら，たとえそれがパソコンのプログラムでも，社会の財産となる．そして，そのプログラムに対して責任がとれる体制が必要になってくるのである．

プログラムには，ほとんどといって良いほどバグがある．またプログラムには色々な要求がついて回る．すなわち，こういう機能をつけて欲しい，等の要求である．パソコンの近くにプログラム作成者がいればその人にたずねればよいが，多くの場合プログラムは使われ続けても作成者は転勤等でいなくなる．また，たとえいたとしても時とともにプログラムの構造を忘れてしまうのである．バグの発生，機能追加の要求に対して対応が困難になるのである．

このようなことを避けるために，プログラム作成者は，必ず作成したプログラムについてのマニュアルを作成しておかねばならない．それは次の二つの内容をまとめたものである．

(1) プログラムの使い方の解説

(2) プログラムの論理構造などの解説

(1)はプログラムの使い方を説明するものであり，市販されているソフトにつけられているので理解されるであろう．(ただし市販されているソフトのマニュアルはほとんどが見にくい．我々は見やすいものを作るべきである．)(2)はプログラムがどのような論理をもち，どのような構成をなしているかを示したものである．これをしっかり記述しておけば，バグの発生や機能追加の要求に対して第三者が容易に対応できるのである．

この(2)のマニュアルの記述法については，産業界や学界で現在とやかくいわれているものであるが，ここではパソコンという規模の小さいシステムについて何を書くべきかを述べよう．それは必ず次のことが含まれるべきである．

(ア) 各モジュール間のインタフェースがしっかり分かること

(イ) 入力および出力のデータの流れが追えること

(ウ) 論理が概論的にも詳論的にも追えること

(エ) データ構造が明らかなこと

この4点がしっかり簡潔に記述されていれば十分であろう．そして，これらの内容がしっかり記述されたマニュアルが備えられて初めて，そのプログラムは社会的財産になるのである．

ここで，一つの例として(2)のマニュアルについての章だてを掲げてみよう．

第1章　本プログラムの目的および概要

第2章　モジュール構成とそれらのインタフェース

第3章　モジュール各論

第4章　データ構造

各章では図，フローチャート，ハイポチャート等を用いて，できる限りの分かりやすさをねらうべきである．

マニュアルの完備していないプログラムは社会の財産にはならない．

Memo　**要求仕様技術**

いかに分かりやすいプログラムを書き，いかにしっかりしたマニュアルを作るかは，ソフトウェアが肥大化した今日非常に重要な課題であり，現実にも学界，産業界で盛んに論じられている．このようなソフトウェアについての技術を**要求仕様技術**などと呼んでいる．

第2章

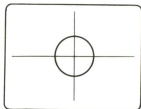
優れたプログラムのコーディング技法

BASICは，大きく二つの特徴をもった言語である．一つは英語と初等数学に大変近い表現方法をとっていることであり，もう一つは翻訳に時間を浪費するということである．この二つの性質を意識すると，我々がBASICでプログラムを記述するとき守らねばならない，いくつかのコーディング技法が生まれてくる．ここではその代表的なものを示そう．

10 英語の文章を書くように

次のプログラムを見てみよう．これは画面中央（640×200ドット）に座標軸を設定し，その原点に円を描くものである．

```
100 LINE (320,0)-(320,199)
110 LINE (0,100)-(639,100)
120 CIRCLE (320,100),100
130 END
```

出力は次のようになる．

このプログラムは数値が多用されていて，いかにも“コンピュータの言葉”のように思われるが，しかしこれはBASICが高級言語であり人間の言葉(特に英語)に近いという特性を十分に生かしていない，と批判されるであろう．320とか100, 639とは一体何なのだ，という疑問がこのプログラムを読む人の心にわいてくる．

上記プログラムを次のプログラムと比較してみよう．

```
100 OX=320:OY=100:' origin=(OX,OY)
110 XMAX=320:YMAX=100
120 LINE (OX,OY-YMAX)-(OX,OY+YMAX)
130 LINE (OX-XMAX,OY)-(OX+XMAX,OY)
140 R=100:' R=radius of circle
150 CIRCLE (OX,OY),R
160 END
```

このプログラムでは，まず数値をイメージの伴う変数名に代入し，その変数名で以下の命令を記述している．このような，ちょっとした工夫でプログラム

は人間に近い言葉となっていくのである．後者のプログラムは，あたかも英語を読むように解説されてゆくであろう．

長いプログラムでは，このような配慮をしているといないとではバグの発見のスピードがまったく違ってくる．そして，もし前者の例のようなプログラムの記述のしかたに固執する人がいるとしたら，その人はBASICを用いずに機械語でプログラミングすることを勧める．BASICの良さは「人間に近い言語である」ということだからである．

実際に我々が注意することをまとめてみよう．

(1) 変数名はできるだけ中味を表現するように．

(2) 注釈文は上手に入れる．

(3) 長い命令は書かない．それは長い文章が読みにくいのと同じである．

(4) 多くのBASIC命令を覚え，分かりやすい命令を選択する．

プログラムは人間の言葉に近づくようコーディングしよう．

Memo　コーディング

コンピュータを利用するとき，我々はコンピュータに作業の手順を示す命令の体系を与えねばならない．これをプログラムと呼んでいる．プログラムは色々な言語で記述される．BASICもその一つの言語である．このようにプログラムをコンピュータの解読する言語で実際に書くことを**コーディング**（coding）という．すなわち，我々のプログラムをコード化するのである．

11 イメージを伴う変数名を

BASIC 言語は，コンピュータに理解される機械語のように非人間的なものではなく，より我々の理解できる親しみやすい言語となるように構成されている．したがって，我々がプログラムを作成するときに，この BASIC 言語の良い点をフルに利用しなければ，BASIC 言語を用いるメリットがなくなってしまう．変数名をどのようにするか，ということもこの考え方に従って決定されなくてはならない．

ほとんどのパソコンに備わる BASIC は，変数名がかなりの長さの文字数になることを許している．(もちろん，先頭の 2 文字しか識別できない，といった制限がつく場合があるが！）我々はこの規約の主旨をしっかりと理解すべきであろう．

〈例〉会社の従業員の給料計算をするプログラムを考えよう．ここでは単純に従業員番号（1 から 50 までとする）を画面に出し，その支給総額と税額とを入力して，差し引きの手取り額を計算し，この明細書を作成するとしよう．
まず次のようなプログラムを考えてみよう．

```
100 'payment calculation
110 FOR K=1 TO 50
120   CLS
130   PRINT "ﾊﾞﾝｺﾞｳ =";K
140   INPUT "ｼﾊﾗｲｶﾞｸ=";A
150   INPUT "ｾﾞｲｷﾝ  =";B
160   '
170   C=A-B
180   '
190   LPRINT "No    ｼﾊﾗｲｶﾞｸ   ｾﾞｲｷﾝ       ﾃﾄﾞﾘ"
200   LPRINT USING "##    ";K;
210   LPRINT USING "########    ";A;
220   LPRINT USING "#######    ";B;
230   LPRINT USING "########";C
240 NEXT K
250 END
```

参考のために一人の従業員の出力結果も示しておこう．

```
No    ｼﾊﾗｲｶﾞｸ      ｾﾞｲｷﾝ       ﾃﾄﾞﾘ
 1    1753420     27534    1725890
```

次に，まったく同一の出力をする，変数名に工夫を入れたプログラムを示そう．

```
100 'payment calculation
110 FOR NO=1 TO 50
120   CLS
130   PRINT "ﾊﾞﾝｺﾞｳ =";NO
140   INPUT "ｼﾊﾗｲｶﾞｸ=";PAY
150   INPUT "ｾﾞｲｷﾝ  =";TAX
160   '
170   INCOME=PAY-TAX
180   '
190   LPRINT "No     ｼﾊﾗｲｶﾞｸ    ｾﾞｲｷﾝ        ﾃﾄﾞﾘ"
200   LPRINT USING "##    ";NO;
210   LPRINT USING "#######    ";PAY;
220   LPRINT USING "######    ";TAX;
230   LPRINT USING "#######";INCOME
240 NEXT NO
250 END
```

この二つのプログラムを比較すれば分かるように，後者の方がはるかに人間の言葉に近づいている．すなわち，変数名を見ればすぐにその変数のプログラム中の役割が理解されるのである．特にプログラムが長くなればなるほど，この恩恵を大きくこうむることになる．

変数名は，その役割りがすぐ分かるように名づけよう．

12 段づけをきちんと

次の左右二つのプログラムは，配列 A (N) に入っている数値の和 S1 および 2 乗の和 S2 を求めるプログラムである．

```
270 S1=0:S2=0                 270 S1=0:S2=0
280 FOR K=1 TO N              280 FOR K=1 TO N
290 S1=S1+A(K)                290   S1=S1+A(K)
300 S2=S2+A(K)*A(K)           300   S2=S2+A(K)*A(K)
310 NEXT K                    310 NEXT K
```

一目見ただけで右側のプログラムの方が見やすいことがよく分かる．このように，BASIC 命令の文体にちょっとした工夫をこらすことで，プログラムはその明快さを変えるのである．

特にこの段づけ技法は，FOR〜NEXT 命令の**入れ子構造**のときに真価を発揮する．いま，配列 A (M, N) に入っている数値のすべての和 S を求めるプログラムを，上例のように段づけしたものと，しないものとに分けて記載してみよう．

```
580 S=0                       580 S=0
590 FOR K=1 TO N              590 FOR K=1 TO N
600 FOR J=1 TO M              600   FOR J=1 TO M
610 S=S+A(K,L)                610     S=S+A(K,L)
620 NEXT J                    620   NEXT J
630 NEXT K                    630 NEXT K
```

FOR〜NEXT の中に FOR〜NEXT があるプログラムは，難解になることが多いが，段づけをしっかりすることで論理が非常に明快になってくるのである．

段づけは FOR〜NEXT だけに応用するものではなく，たとえば次のように IF 命令の中でも使える．

```
140 INPUT X
150 IF X=1 THEN Y=X*X ELSE Y=X*X+2*X+3
```

これは入力された X の値が 1 なら Y に X^2 を，$X \neq 1$ なら X^2+2X+3 を入れるプログラムであるが，これを次のように書いてみると明快になる．

```
140 INPUT X
150 IF X=1 THEN Y=X*X
        ELSE Y=X*X+2*X+3
```

また次の例のように，IF 命令で論理を分類したとき，その分類された論理のまとまりを表わすのにも段づけが用いられる．

```
270 IF D<0 THEN *IMAGE
280   X1=B+SQR(D):X1$=STR$(X1/2/A)
290   X2=B-SQR(D):X2$=STR$(X2/2/A)
300   GOTO *INSATU
310 *IMAGE
320   X1$=STR$(B/2/A)+"+"+STR$(SQR(-D)/2/A)
330   X2$=STR$(B/2/A)+"-"+STR$(SQR(-D)/2/A)
340 *INSATU
350 PRINT X1$:IF X1$<>X2$ THEN PRINT X2$
```

これは2次方程式の解を求めるプログラムであるが，行番号のラベルを区切りとして一つの論理的なまとまりを段づけすると，非常に見やすくなることを理解してもらいたい．

以上の三つの例で段づけを説明したが，このような簡明な工夫でプログラムが生き生きしてくることに我々は気をつけるべきであろう．

段づけなどを利用して論理をみやすく整然と!!

Memo フリーフォーマット

昔の FORTRAN などは，この節の段づけを許さなかった．しかし，近年の高級コンピュータ言語は，1行中にどのように命令を書いてもそれを許してくれる．この特徴をフリーマットと呼ぶことがある．

13 行番号は1000番から

最初に次のプログラムを眺めて見よう，これはDATA文中にあるデータの平均と標準偏差とを求めるものである．

```
10 N=10:DIM A(N)
20 FOR K=1 TO N
30   READ A(K)
40 NEXT K
50 DATA 10,6,8,3,7,4,1,4,6,9
60 '
70 S1=0:S2=0
80 FOR K=1 TO N
90   S1=S1+A(K)
100    S2=S2+A(K)*A(K)
110 NEXT K
120 M=S1/N:V=SQR(S2/N-M*M)
130 PRINT "m=";M;"   v=";V
140 END
```

プログラムの行番号は，この例のように10おきにとるのが普通である．もちろんこの間隔の値は，次のRENUM命令の第3パラメータで容易に変更が可能である．

```
RENUM N1, N2, S
```

このSの値に間隔としてとりたい値を代入すればよい．通常間隔を10にとるのは，追加・訂正のしやすさと見やすさが理由である．

ここで再び上記プログラムを眺めて気がつくのは，行番号90から行番号100にゆくところで，プログラム作成者としては不本意な段差が生じるということである．このことは，文番号が990から1000にいくとき，および9990から10000にいくときにも起こることである．このような不本意な段差が，FOR～NEXTなどできれいに段づけしているところで発生すると，せっかくの見やすさに傷がついてしまう．

我々は通常行番号を1000から始める．短いプログラムなら100から始めてもよい．長いプログラムなら当然10000から始めてもよい．しかし，あまり行番号の桁が大きいと，見にくくなることは避けられない．通常のBASICで記述さ

れるプログラムでは，だいたい1000番から始めると不都合がないのである．

本節のプログラムは短いので，行番号を100からとることにする．このためには，次の命令を実行すればよい．

```
RENUM  100, 10, 10
```

その実行結果を表示してみよう．ずい分とプログラム・リストが引き締まって見えることに注意してもらいたい．

```
100 N=10:DIM A(N)
110 FOR K=1 TO N
120   READ A(K)
130 NEXT K
140 DATA 10,6,8,3,7,4,1,4,6,9
150 '
160 S1=0:S2=0
170 FOR K=1 TO N
180   S1=S1+A(K)
190   S2=S2+A(K)*A(K)
200 NEXT K
210 M=S1/N:V=SQR(S2/N-M*M)
220 PRINT "m=";M;"    v=";V
230 END
```

また，モジュールの先頭となる番号は区切りのよい行番号にしておくことをすすめる．こうすることで，一目で処理の区切りを理解できるのである．

行番号の工夫だけで，プログラムはずい分と明快になる．

Memo 行番号が足りないとき

行番号の最大値は，約65,000(2バイトで表わせる最大数)である．10行間隔で番号をつけると，6,500ステップ以上のプログラムは作れないことになる．しかし，それを心配する必要はない．大きなプログラムは，分割しそれをCHAIN命令で結合するのが常識だからである．

14 GOTOをなくそう

プログラミング技法の最も有名なものの一つに「GOTO をなくせ」というものがある．次のプログラム例を見てみよう．

```
270 S=0:N=1
280 IF N>100 THEN 320
290 S=S+N
300 N=N+1
310 GOTO 280
320
```

これは 1 から 100 までの自然数の和を求めている．これを FOR～NEXT 命令で書き換えると次のようになる．

```
270 S=0
280 FOR K=1 TO 100
290    S=S+K
300 NEXT K
```

これは極端な例ではあるが，GOTO 命令を用いないプログラムがいかに簡潔になり見やすくなるかを良く示している．

我々は，文章を 1 行 1 行読むような考え方に慣れ親しんでいるため，プログラムを追うとき論理がジャンプするような発想になじめないのである．そのためか，多くのバグが GOTO 命令の回りに発生するといわれている．また GOTO 命令のために，修正がしにくいことがしばしばあるともいわれる．

GOTO 命令をなくす代表的な方法として，上記のように FOR～NEXT 命令での代用がある．またパソコンの上位機種には，WHILE～WEND という命令が備えられている．次の例を見てみよう．

```
100 S=0:N=0
110 IF S>100 THEN 150
120   N=N+1
130   S=S+N*N
140 GOTO 110
150 PRINT N
160 END
```

これは自然数の 2 乗の和

$1^2+2^2+3^2+\cdots\cdots+N^2$

で，その和が初めて100を越えるNを求めるプログラムである．これはWHILE～WENDを用いて次のように書き換えられる．

```
100 S=0:N=0
110 WHILE S<=100
120   N=N+1
130   S=S+N*N
140 WEND
150 PRINT N
160 END
```

このWHILE～WENDも上手に用いると，プログラムを非常に分かりやすくしてくれるものである．

その他，GOTO文を減らす手法として

```
ON K GOTO N1, N2, ……
```

という命令を用いるとか，条件のテーブル化（18節参照）などが有名である．

GOTO命令をなるべく用いないよう工夫しよう．

Memo　X=A : IF……THEN　X=B

初心者のプログラムにしばしば見られるのが，次のようなコーディングである．

```
250 IF S>0, THEN X=A : GOTO 270
260 IF S<=0 THEN X=B
270 '
```

これは明らかに次の1行にまとめるべきである．

```
250 X=A : IF S<=0 THEN X=B
```

15 注釈文は丁寧に

高級言語であるBASICによるプログラミングは，その命令体系自体英語に近く，人間に近い表現となる．しかし，いくらBASICが人間に近い言語であるからといっても，やはりそこには限界がある．我々はその限界をできるかぎり小さくすべきであるが，その手段の一つとして注釈文の挿入というものがある．

注釈文とは，REMまたは'（アポロストロフィ）で始まる文であるが，これはプログラムの実行には関係せず，プログラム内に注釈をつける目的にのみ利用される．次の例を見よう．

```
270 GOSUB 700
 :
 :
700 LPRINT A
710 LPRINT B
 :
850 RETURN
```

```
270 GOSUB 700
 :
 :
700 'ケイサンケッカ ノ インサツ
710 LPRINT A
720 LPRINT B
 :
860 RETURN
```

右側は左側のプログラムでサブルーチンの先頭番号を注釈文にし，そのサブルーチンの内容を表示したものである．こうすることで，我々プログラムを読む者は安心してサブルーチンの内容を解読しにかかれるのである．

このように，注釈文はプログラムを読みやすくしてくれる．したがって，これを丁寧にプログラム内に挿入していくべきであるが，しかし冗長な注釈を多く入れることは，かえってプログラムを見にくくしてしまう．簡潔で要領の得た注釈を施すべきである．

プログラムには他人が読んでも分かるようなきちんとした注釈文を入れよう．

16 1行1命令が原則

BASIC では同一文番号中にコロン：を区切り文字として，複数の命令を記述することを許している．

〈例1〉　170　S2=0 : FOR K=2 TO 5 : S2=S2+A(K) * A(K) : NEXT　K

後述（85節を参照）する通り，この記法はそれなりのメリットがあるので一概に非難することはできないが，原則としては避けるべきであろう．

〈例1〉については，通常次のように記すべきである．

```
170 S2=0
180 FOR K=2 TO 5
190    S2=S2+A(K)*A(K)
200 NEXT K
```

その理由の一つは，**マルチステートメント**（例1のような記法をいう）を多用するとプログラムが横に長くなり，論理の流れが見えにくくなることである．またもう一つの理由は追加・訂正がしにくくなることである．したがって，この二つの理由を承知の上でマルチステートメントを用いることは，いっこうにさしつかえないであろう．例として次のコーディングを見よう．

〈例2〉

```
170 S2=0:K1=2:K2=5
180 FOR K=K1 TO K2
190    S2=S2+A(K)*A(K)
200 NEXT K
```

この例2の行番号170は初期値の設定であるから，かえってマルチステートメントを用いた方が見やすくなる．

プログラミングも語学と同じで，例外のない原則というものはない．したがって我々は時と場合によって原則を使い分けることが大切である．しかし，それは原則を意識した上でなされるべきである．マルチステートメントもその原則をしっかりと認めた上で使用してもらいたい．

横にダラダラと長いコーディングをしてはならない．

17 コンパイラのコーディング技法は通用しない

FORTRAN などのコンパイラ言語に慣れ親しんだ人は，つい BASIC でもコンパイラ言語で用いた技法がそのまま通用するものと思い込んで，その技法を使ってしまう．しかし，BASIC というインタプリタ言語とコンパイラ言語は，基本的に異なり，したがってコンパイラ言語で通用したことが BASIC では通用しないことがある．ここではその有名なものをいくつか示してみよう．

(1) “簡単な関数は呼び出すな”は本当か？

FORTRAN などに慣れた人は BASIC 命令

```
200  X=ABS (A)
```

を次のように書いてしまう．

```
200  X=A : IF A<0  THEN  X=-A
```

この二つの表現は同じだが，コンパイラ言語では下の命令の方が計算スピードが速い．しかし，BASIC プログラムにおいては，上の命令の方が速いのである．それは BASIC は翻訳に手間どるから実行時間の節約よりも翻訳時間の節約の方が大切だからである．

同じことが，簡単な指数計算についてもいえる．コンパイラに慣れた人は，次の BASIC 命令

```
500  Y=X^3
```

を次のように書いてしまう．

```
500  Y=X * X * X
```

しかし，上と同じ理由で上の命令の方が BASIC 言語に適している．

(2) 2 * A か A+A か

FORTRAN に慣れた人は BASIC 命令

```
700  S=2 * A
```

を次のように書く場合がある．

```
700  S=A+A
```

これはコンパイラ言語では「乗法より加法を用いよ」というコーディング技法があるからである．すなわち，CPUの処理速度は乗法より加法の方が速いのである．それなら当然BASICでもこのことはいえることになる．しかし，前にも述べたようにBASICは翻訳が処理時間の中で大きなウエイトを占めており，その中で“＊か＋”という議論は色あせてしまう．すなわちコンパイラは翻訳さえ終われば，後はその翻訳された実行プログラムの実行時間だけが問題になり，必然的に＊計算よりも＋計算の方が速い，ということは大切になる．これに対して，BASICは一つひとつのBASIC命令を常に解読（翻訳）しながら実行している．当然解読の方が命令実行よりもはるかに時間がかかってしまう．このとき，“＋か＊”は意味のない議論となってしまう．BASICでは以上のことから，細かな実行速度の大小の議論よりもプログラムの見やすさの方に重点を置いたコーディングをした方が良いことになる．

以上二つの例から分かるように，コンパイラ言語とインタプリタ言語とではその翻訳のしかたの違いからくる性格のため，色々と違った計算技法が対応する．この性格の違いをしっかり見抜いておかないと，まったく無意味な計算技法を振り回してプログラムを見にくくし，無用な煩雑さを招くことになってしまう．

BASIC言語の特徴をしっかりつかんでコーディングしよう．

Memo **BASICのコーディング技法の基本**

この節で示したように，とにかくBASICは翻訳に手間どる．したがって，BASICプログラムの処理効率を上げる一つの大きな原則はBASIC命令の数を少なくすることである．そのために，BASICは多彩な命令を用意してくれているので，それらを上手に使いこなしてゆくべきである．

18 条件文の乱立は避けよう

次のプログラムを見てみよう．

```
300 IF K=1 THEN S=100
310 IF K=2 THEN S=300
320 IF K=3 THEN S=150
```

このような記述のしかたは見やすいようだがどうも美しくないし，効率が悪い．これは次のように書くべきである．

```
300 A(1)=100:A(2)=300:A(3)=150
310 S=A(K)
```

こうすることで処理速度は向上し，修正等も楽に行える．そしてプログラム自体引き締まる．

このように，条件が重なるとき，配列を用いると上手にそれらをまとめることができる場合が多いことは銘記しておくべきである．

また次の例を見てみよう．

```
400 IF D<0 THEN 450
410 IF D=0 THEN 490
420 X1=(-B+SQR(D))/2/A:X2=(-B-SQR(D))/2/A
430 PRINT X1,X2
440 GOTO 500
450 S1$=STR$(-B/2/A):S2$=STR$(SQR(-D)/2/A)
460 X1$=S1$+"+"+S2$:X2$=S1$+"-"+S2$
470 PRINT X1$,X2$
480 GOTO 500
490 PRINT -B/2/A
500 '
```

これは 2 次方程式 $ax^2+bx+c=0$ の解を求めるプログラムであるが，IF 命令と GOTO 命令とが入り乱れて難解である．これは次のようにすべきであろう．

```
400 ON SGN(D) THEN 440,480
410   ' D>0
420   X1=(-B+SQR(D))/2/A:X2$=(-B-SQR(D))/2/A
430   PRINT X1,X2:GOTO 500
440   ' D<0
450   S1$=STR$(-B/2/A):S2$=STR$(SQR(-D)/2/A)
```

```
460     X1$=S1$+"+"+S2$:X2$=S1$+"-"+S2$
470     PRINT X1$,X2$:GOTO 500
480     ' D=0
490     PRINT -B/2/A
500     '
```

すなわち，分岐するところを1箇所に集中することでプログラムを読む人の理解を助けることができるのである．このON～GOTO命令に似たものとしてON～GOSUB命令があるが，これらの命令を用いることで多くのIF命令がプログラム中に乱立しないよう注意すべきである．

たくさんのIF命令があると頭が混乱する．しっかりと条件を整理しよう

Memo **ON～GOTO命令の使用上の注意**

ON K GOTO N1, N2, N3, ……, Nm

において，Kが1からmの間の自然数をとるときに，N1，N2，……で示された行番号に分岐するのが，この表題の命令であるが，Kが0もしくはmより大きい値のときには，この命令は何もしない(K<0のときはエラーとなる)．したがって万一プログラムにバグがあり，K=0とかK>mの値をとると，この部分はそのまま正常に実行され，バグがなかなか見つけられなくなることがあるのに注意しよう．対策としては99節参照．

19 イメージを伴う数値を用いよう

640×200 ドットのグラフィック画面に関数 $y=x^2$ のグラフを描いてみよう．

```
10 DEF FNF(X)=X*X
20 FOR X=-10 TO 10 STEP .1
30   Y=FNF(X)
40   XG=320+20*X:YG=100-10*Y
50   IF YG<0 OR YG>199 THEN 70
60   PSET (XG,YG)
70 NEXT X
80 END
```

この行番号 50 において 199 という数値が現われている．これは，640×200 ドットの画面では次のように番地がつけられているからである．

横方向に 0〜639

縦方向に 0〜199

この縦番地の限界をチェックしたのが行番号 50 なのである．このステップで 199 という数を用いた理由は理解されるが，やはり我々は次のように記述すべきだろう．

```
50 IF YG<0 OR YG>=200 THEN 70
```

プログラムを読む人は 200 という数値で画面の上限チェックしていることを理解できるのである．似た例として次のようなものがある．

```
10 WIDTH 80,20
20 INPUT "x,y=";X,Y
30 IF X<0 OR X>79 THEN 20
40 IF Y<0 OR Y>19 THEN 20
50 LOCATE X,Y:PRINT "A"
60 END
```

これは入力された X, Y の値を座標とするテキスト画面上の場所に A という文字を表示するプログラムである．行番号 30, 40 はやはり次のようにすべきであろう．

```
30 IF X<0 OR X>=80 THEN 20
40 IF Y<0 OR Y>=20 THEN 20
```

こうすることで行番号10のWIDTHのパラメータ値と対応をとることができる．もう一つの例を示そう．

```
10 INPUT "code=";C
20 IF C<0 OR C>255 THEN 10
30 C$=CHR$(C)
40 PRINT C$
50 END
```

これは，入力された値Cをコードとしてもつ文字を出力するプログラムであるがこの行番号20の255という値も異様な感を与える．入力されたコード値が，0と255の間に入っていなければならないことをチェックしているわけであるが，この行番号20は，次のどちらかにすべきである．

```
20 IF C<0 OR C>=256 THEN 10
20 IF C<0 OR C>&HFF THEN 10
```

こうすることで，入力されたCの値が16進2桁(すなわち1バイト)に納まっているかをチェックしているのだな，とすぐに理解できるのである．

プログラムに記述する数値はイメージが伴う値を採用しよう．

Memo　長い命令への対応

CIRCLE命令などはオペランド部のパラメータが多く1行に一つの命令が納まらないことがある．

(例)　700 CIRCLE (CENTERX, CENTERY), RADIUS, COLOR1, RAD1, RAD2, RATIO, F

このようなときには下のコーディング例のように分かりやすさの心配りが必要であろう．

```
700 CIRCLE (CENTERX, CENTERY), RADIUS, COLOR1, RAD1,
                                  RAD2, RATIO, F
```

20 定数にも変数名を

640×200 ドットのディスプレイ画面に，下記のような円グラフを表示するプログラムを考えてみよう．(この CIRCLE 命令の使い方については 65 節参照．)

これについては次のようなプログラミングが可能である．

```
100 CLS 2
110 P=3.14159:T0=5*P/2
120 FOR K=1 TO 5
130   READ W:DT=2*P*W/100
140   T1=T0:IF T0>2*P THEN T1=T0-2*P
150   CIRCLE (320,100),100,,-T0,-(T0-DT)
160   T0=T0-DT
170 NEXT K
180 DATA 30,25,20,15,10
190 END
```

しかし，プログラムをこのように定数を用いて記述すると後で大変である．というのは，これを修正しようとするとき，プログラム全体にわたって再校正せねばならなくなるからである．このプログラムは画面中央に円グラフの中心をとり，半径を 100 ドットにしたが，もしこれを画面の右側によせもう少し小さな円にしたい，などと思ったとき修正はいちいちプログラムの中味にまで及んでしまう．この例は短いプログラムであるから修正は容易だが，もっと大きく複雑なものでは，その修正が容易でないことは想像できるであろう．

定数を用いたプログラムの修正の困難さに対処する手段として，それらの定数に変数名を割り振る方法があげられる．上記プログラムは次のようにコーディングすべきなのである．

```
100 CLS 2
110 N=5:P=3.14159:TO=5*P/2
120 CX=320:CY=100:R=100
130 FOR K=1 TO N
140   READ W:DT=2*P*W/100
150   T1=TO:IF TO>2*P THEN T1=TO-2*P
160   CIRCLE (CX,CY),R,,-TO,-(TO-DT)
170   TO=TO-DT
180 NEXT K
190 DATA 30,25,20,15,10
200 END
```

こうすることで，プログラムの先頭行に定義されている変数CX，CY，Rの値を変えるだけで，プログラムの修正は完了するのである．長いプログラムであれば，このような手法をとらない限り，プログラムの修正はそのプログラムの長さに比例した時間と手間ひまを要することになる．

定数に変数を割り振る方法のメリットとして，この節の論拠以外にも10節で述べたようにプログラムの見やすさということがあげられる．またBASICの構造上，一般的に計算スピードが速くなるということもメリットの一つに数え上げて良いであろう．

実数を直接プログラム中に入れると変更が大変になることを覚悟しよう．

Memo　定数への変数名のつけ方

定数に割り振る変数名は当然その定数を印象づけるものであるべきである．たとえば

円の中心	(CX，CY)	(Center of X(Y) co-ordinate)
半径	R	(Radius)
円周率	PAI，P	

このような努力をしないと変数名を覚えるだけで一苦労となる．

21 10進か16進か

我々の日常生活においては，16 進数を扱うことはない．したがって，プログラムを記述するとき，できることなら 16 進数を表面に出すべきではない．しかし，コンピュータの内部で 2 進法の計算が実行されている以上，その親戚である 16 進数でプログラムを記述した方が，かえって分かりやすいということがよくある．

たとえば画面消去の命令を

```
PRINT  CHR$(12)
```

というように書くよりも，

```
PRINT  CHR$(&H0C)
```

と書いた方がよい場合が多い．(もちろん，多くのパソコンではこの命令は CLS というように，制御コードを記述しないですむようにしている．) その理由は，このような制御コードをまとめた表は，ほとんどの場合 16 進数で示されているからである．

同じ理由から文字コードも 16 進で扱う方が分かりやすい．次のコーディングは文字変数 S$に入っている一つの小文字のアルファベットを大文字に直し，文字変数 C$に入れるものである．

```
270  S=ASC(S$)
280  C$=CHR$(S-32)
```

アスキーコードでは小文字のコードから 32 を引くと大文字のコードになるという規約を用いたものである．しかしコード表を見るとき 32 という値は一度 16 進の

```
&H20
```

に直しておかないと，我々は確認することができないのである．したがって，次のようにコーディングした方が親切である．

```
270  S=ASC(S$)
280  C$=CHR$(S-&H20)
```

また，我々はしばしば整数型変数に割り振られた2バイトの数値表現領域の各ビットに意味をもたせることが多い．それはパソコンのメモリ容量が小さいためにメモリを節約するためにするのである(85節参照)．このとき，たとえば整数型変数Xの第8ビットがON(すなわち1)かどうかを調べるときに，次のような10進表現を用いると非常に分かりにくくなる．

```
IF X AND 128=1 THEN~
```

128とは一体何か，と頭をかしげてしまう．やはり次のように16進で記述しなければならない．

```
IF X AND &H80=1 THEN~
```

MSXパソコンのように2進数を表現できるパソコンでは，もっと丁寧に次のように書くべきである．

```
IF X AND &B10000000=1 THEN~
```

以上のように，メモリとか制御コードとかいったハードウェアに近いデータ表現をするときには，計算のしやすい16進数（あるいは2進数）を用いるべきである．

Memo 16進数

16進数&Huvwxは次のように10進数Nに変換できる．

$$N=u\times16^3+v\times16^2+w\times16+x$$

ここでu，v，w，xは0～F（10進で0～15）までの数である．

コンピュータの世界では16進数がよく用いられる．本来は，ビット単位に構成されているのであるから，2進数の方がコンピュータにはふさわしいのかも知れないが，その親戚である16進数の方が，我々が日常用いている10進数に近いという理由と，1バイトが16ビットであるというハードウェア上の理由から，我々は16進数を多用するのである．

22 一時的変数はなるべく用いない

我々はプログラムを読むとき，必ず変数に付随する役割を頭に覚えているものである．しかし大きなプログラムになると必然的に多くの変数名が出てきて，それらの役割をいちいち覚えるのは大変になってくる．したがって，プログラムに現われる変数の数は，少なければ少ないほどそのプログラムは読みやすくなるものである．いま次の論理を考えよう．

〈例〉　変数 X の値を 2 乗したものを，新たに変数 X に代入する．しかし，もしその新たな X の値が 100 を超えていたなら，もとの X の値に変数 X の値をもどす．これを文章通りにプログラミングすると，不可避的にその場限りの一時的変数が現われてしまう．（ここではその変数名を A とする）

```
260  A=X
270  X=X * X
280  IF  X>100  THEN  X=A
```

現実にはこんなプログラミングをする人はいないと思う．すなわち，次のように 1 ステップで表現できる．

```
260  IF  X * X  <=100  THEN  X=X * X
```

論理が込み入ると，上例のように簡単には一時的変数を消却できないことが多い．しかし，プログラムを読む立場になって考えれば，このようなその場限りの一時的な変数は努力して用いないようにすべきである．

上例のように論理を工夫することで一時的変数を除くことができる場合と，BASIC 文法をしっかりマスターすることで一時的変数を回避できる場合とがある．後者では有名な SWAP 命令を例示してみよう．

2 変数 A, B の値を入れ換えるには，どうしてもその場限りの変数が必要になる（ここではそれを C とおく）．これを用いて次のように記述できる．

```
100  C=A
110  A=B
```

```
120  B=C
```

BASICのマニュアルにしっかり目を通していれば，BASICには大変便利な命令があることに気づく．それがSWAP命令である．これを用いると上の3行は次の1行ですませられる．

```
100  SWAP  A, B
```

このようにBASICをしっかりマスターすることでも一時的な変数の利用を避けることができるのである．しかし例外もある．次の例を見てみよう．

```
250  IF  B＊B−4＊A＊C>0  THEN  S=2
260  IF  B＊B−4＊A＊C=0  THEN  S=1
270  IF  B＊B−4＊A＊C<0  THEN  S=0
```

この例では当然次のようにすべきである．

```
250  D=B＊B−4＊A＊CS=0
260  IF  D>0  THEN  S=2
270  IF  D=0  THEN  S=1
```

こうすることで処理速度は向上し，またキーインミスも少なくなる．
以上の例外を認めた上で，我々は次のことを主張する．

一時的にしか用いない変数は，なるべく用いないようにしよう．

Memo **SWAP命令**

BASICの命令の中でこの命令はぜひ覚えておくべきである．それは二つの変数の入れ換えという操作をよく用いるからであり，それを本節で述べたように三つのステップで行ったのでは大変処理効率が落ちてしまう．特にソートなどをパソコンで行うとき，この命令を知っているかどうかで処理時間に少なからぬ影響が出てしまう．

23 IF～THEN～ELSE～は避けよう

BASIC は条件判定として次の構文を用意している．

IF～THEN～ELSE～

有効であれば，この命令を当然存分に用いてよいのだが，やはりそれなりの心配りが必要である．

次のプログラムを見てみよう．これは任意の数値 A に対して，その平方根（A<0のときは $\sqrt{|A|}\,i$）を求めるものである．

```
100 INPUT "A=";A
110 IF A>=0 THEN X=A ELSE X=-A
120 X=SQR(X)
130 PRINT "root A=";X;
140 IF A<0 THEN PRINT "i" ELSE PRINT
```

これはこれで正常に動くが，次のプログラムと比較してみよう．

```
100 INPUT "A=";A
110 X=A:IF A<0 THEN X=-A
120 X=SQR(X)
130 PRINT "root A=";X;
140 IF A<0 THEN PRINT "i";
150 PRINT
```

我々は後者の方を勧める．すなわち，同じ内容を示すなら，一文はできるだけ短い方がよい．IF～THEN～ELSE～とすると，文章的には複文となり冗長となる．すなわち，

もし～ならば～であり，そうでなければ～である

というのは長い．たとえば，上例の二つの行番号 110 を見てみよう．

```
110 IF A>=0 THEN X=A ELSE X=-A
110 X=A : IF A<0 THEN X=-A
```

前者は複文構造となっているが，後者は二つの簡単な文章の連結である．明らかに後者の方が分かりやすい．

IF～THEN～ELSE～の構文は多くの場合，もっと短く簡単な命令の和に置

き換えられる，我々は長い一文より簡潔な二文の方を解しやすいと感じる．このことに気をつけながらELSEを用いてもらいたい．

ELSEはなるべく用いるな

複文を避けよ，という主旨は次節のIF〜THEN〜IFはやめようということにつながる．これをもっと一般化して，長い冗長な一命令を避けよう，とも換言できる．次の二つの例を見てみよう．

(例)
```
250 PSET (Y-X*SIN(T),Z-X*COS(T))
```

(例)
```
250 XG=Y-X*SIN(T):YG=Z-X*COS(T)
260 PSET (XG,YG)
```

これは3次元座標（X，Y，Z）を画面に描くものだが，やはり下の例の方が見やすく，修正もしやすいであろう．

長い冗長な命令は短かい簡潔な命令に分割しよう．

Memo **ELSEの消去**

本節の例でもそうだが，ELSEを消そうとすると一時的に用いる変数を用いざるを得ないことが多い．（本節のプログラム例では変数Xがそれである.）22節で述べたように1次的な変数はなるべく用いない方がよい．そうすると，我々は二つの原理の間にはさまれて困惑する．このような場合は我々の判断を素直に入れるべきであろう．すなわち，どちらの方が分かりやすく効率が良いか，を自分で決めるのである．

24 IF～THEN IF～はやめよう

多くのパソコンの BASIC は，IF～THEN に続く命令に IF を用いて良いように作られている．

〈例1〉

```
100 IF A>0 THEN IF B>0 THEN X=1 ELSE X=0
```

この例からも分かるように IF～THEN IF～という構文は非常に見にくいものであり，回避すべきプログラミングである．

この例は次のようにすべきであろう．

```
100 IF A>0 AND B>0 THEN X=1
110 IF A>0 AND B<=0 THEN X=0
```

常にこのように簡単に書き換えられるものでもないが，論理を工夫することで必ず上例1の構文は避けられるはずである．たとえば次のような表を作ることで，このことが実現しやすくなる．

A	B	X
＋	＋	X＝1
＋	－or 0	X＝0
－or 0	＋	NOP
－or 0	－or 0	

ここで NOP とは何もしない (no operation) のことである．このような表のことを**ディシジョン・テーブル** (decidion table) と呼び，条件判定が多い場合に論理を整理するのに役立つものである．

以上のように，何らかの工夫をすることで冗長な IF～THEN IF～構文は回避することが望ましい．1行の命令が長ければ長いほど，読む人の理解を苦し

めることになる．

IF～THEN IF～は論理を整理して複数行に分けよう．

Memo　構造化プログラミング

プログラムの作成のしかたへの要請として本書１節では(1)～(5)を掲げたが，これらをいかに実現するかについては色々な説がある．その中で有名なものとして表題のものがある．これは次のような主張から成り立っている．

(a)　モジュール化を徹底する

(b)　GOTO 文をなくす

(c)　段づけをしっかりする

⋮

近年パソコンのプログラムにも以上のことが要求されているのである．

Memo　GOTO と飛び出しは REM 文に

GOTO 命令などで飛ぶとき，我々はその先の文番号には注釈文をあてるべきである．

```
730 GOTO 950
 :
950 ' リソク ノ ケイサン
```

こうすることで二つのメリットが生まれる．一つはプログラムが見やすくなることであり，もう一つは訂正・追加がしやすくなるのである．

25 計算は計算機に

次のコーディングを見てみよう．

```
270  V=4.18879 * R^3
```

これを見てすぐには何の計算をしているかは不明である．これに対して次のコーディングを見てみよう．

```
270  P=3.141592 : V=4 * P/3 * R^3
```

この形なら我々はすぐに体積の公式 $V=\frac{4}{3}\pi r^3$ を思い浮かべることができる．

もう一つの例を上げよう．

```
140  Y=.434294 * LOG(X)
```

やはり読む人は何をいっているか分からない．これに対して次のように記述してみよう．

```
140  Y=LOG(X)/LOG(10)
```

数学に少し慣れた人ならすぐに“これは常用対数の計算だ”と気づく．

この二つの例は処理効率を焦るあまりプログラムの分かりやすさを犠牲にした例である．コンピュータに $4\pi/3$ を計算させないために電卓でそれを計算し，4.18879 といった見なれない数値をコーディングしてしまったのである．LOG についても同様である．

我々は，1ミリセカンドの時間を問題にするようなプログラムの作成をするときは除いて，プログラムの分かりやすさを絶対に放棄してはいけない．電卓で計算させるならマイコンにそれをさせればよい．

計算はコンピュータに任せよう．

26 変数の初期値設定はしっかりと

次のプログラムを考えよう．

```
100 S=0
110 FOR K=1 TO 100
120   S=S+K
130 NEXT K
140 PRINT S
150 END
```

これは1+2+……+99+100を求めるプログラムである．パソコンのRUNコマンド機能を知っていると行番号100は不要である．というのはRUNが実行されると，すべての数値変数は値が0に設定されるからである．しかし我々はプログラムを書くとき，必ず行番号100のような変数の初期設定をしっかりする習慣をつけることを要求する．

プログラムが複雑になると色々なところから，ある部分が呼ばれる．そのとき上例ではSという変数に100という数値が入って，この部分が呼ばれることもあるわけである．Sという変数は他では使っていないという自信があるかも知れない．またこの部分は絶対に他から呼ばれないと確信しているかも知れない．しかし，プログラムは時間とともに変更・修正されていくものである．後になって，この部分が呼ばれ，Sという変数が呼ぶ側のルーチンで使われるような変更がなされるかも知れない．たかだか1ステップをさぼってそのような危険を招くようなことはすべきでない．

変数の初期値設定において次のことを知っていると便利である．

(1) DIMの宣言の直後にはその宣言された変数が数値変数なら0，文字変数ならヌルストリングが入る．

(2) DEFINT等の宣言後にはその宣言された頭文字をもつ変数は0（文字変数宣言ならヌルストリング）が入る．

変数の初期値設定はこまめにていねいにする習慣を．

27 命令は簡潔に

BASICはインタプリタ言語である．コンパイラ言語(FORTRAN，COBOLなど）と違って命令の解読に非常に時間がかかる言語である．したがってコーディングに際し，1ステップでもBASIC命令が少ないことは処理時間を短縮するのに大きな寄与をするものである．そのためには，論理を工夫して不要な命令を書かないこと，および簡潔なBASIC命令を選択することが重要である．

論理を工夫して不要な命令を書かない，ということは我々は常にプログラミングの際，心に留めておかねばならないことであり，これはあらゆる言語に共通していることである．しかし簡潔なBASIC命令を選択するということは，他の言語にもまして重要となる．たとえばFORTRAN（あるいはCOBOL）で

```
A=B+C
Y=X+A
```

という2ステップの命令を書くことと

```
Y=X+B+C
```

と，1ステップで書くことではそれほど実行処理時間に違いはない．これに対してBASICで

```
250 A=B+C
260 Y=X+A
```

と書くことと，これを1行にまとめて

```
250 Y=X+B+C
```

と書くことでは処理時間に大きな差がでる．

この例でわかるように，BASICのインタプリタ言語としての特徴をしっかりつかみ，コーディング時に命令を短く書く努力をすべきである．

BASICを用いてのコーディングでは冗長は許されない．

BASICは，コーディングの冗長さをなくせるよう，他の言語に比べて色々な種類の命令を用意してくれている．したがって，パソコンのプログラミングに

はできるだけ多くの命令を頭に入れておくことが大切である．たとえば，22節で述べたSWAP命令がよい例となる．

```
SWAP  A, B
```

この命令を知らないと，次のように記述しなければならなくなる．

```
X=A : A=B : B=A
```

ずいぶんに分かりにくい．もう一つの例として次の左右の命令を見てみよう．

```
A$=MID$ (X$, LEN (X$)-2, 2), A$=RIGHT$ (X$, 2)
```

これらは文字X $の左側の2文字を変数A $に代入する命令であるが，明らかに右の方が簡潔で分かりやすい．

Memo 整数型・実数型

数値を表現するしかたとして，BASICは表題の2通りの方法を用意している．整数型はまさに整数のみを表現し，2バイトの長さの領域内で表わされる．すなわち−32768から32767までの整数を表現する．これに対して実数型とは小数×10 ^{n}の形の数を表現する．これには単精度型と倍精度型とがあり，小数部の有効桁が各々7桁，16桁となっている．この実数型のバイト数としてBASICは単整度型に4バイトを，倍精度型に8バイトをあてている．当然整数形，単精度実数形，倍精度実数形の順でパソコン内のメモリ消費が大きく，またそのことと関連して計算スピードも遅くなることになる．

28 BASIC命令に精通しよう

実行する命令の数を減らし，一つの命令についても不用なパラメータはつけないでおくことは，BASIC のように翻訳に時間のかかる言語では特に大切なことである．このことを実現するいくつかの方法をここで紹介するが，これらの例によって主旨を理解してもらいたい．

〈例 1〉 配列変数 S(1)〜(10)を定義し，これらを 0 に初期設定せよ．

この例に対して次のコーディングが考えられる．

```
10 DIM S(10)
20 FOR K=1 TO 10 : S(K)=0 : NEXT K
```

しかし，BASIC に精通していれば，この 2 行の命令の一方は不要であることが分かる．すなわち，BASIC は 10 までの配列には DIM 宣言は不要とし，また DIM 宣言後に最初に用いられる配列の値を 0 に初期設定しているのである．特に後者についてはしっかりと覚えておくと便利である．たとえばプログラム途中 100 個の要素からなる配列 A について，その中味をクリアしたいときには

```
FOR K=0 TO 1000 : A(K)=0 : NEXT K
```

というコーディングをすると時間を浪費する．次のようにする．

```
ERASE A : DIM A(100)
```

両者とも 1 行の命令であるが，前者は 100 個の BASIC 命令を実行し，後者は二つの実行ですんでいるのである．

〈例 2〉 $0.1^2+0.2^2+\cdots\cdots+1.0^2$ の和を求めるプログラムを作ろう．

FORTRAN などに慣れている人は，しばしば次のようにコーディングしてしまう．

```
100 S=0
110 FOR K=1 TO 10
120   S=S+(K*.1)^2
130 NEXT K
```

これは，BASICの多彩な機能を覚えていないことからくる冗長性である．これは，次のようにすべきだろう．

```
100 S=0
110 FOR X=0 TO 1.01 STEP .1
120   S=S+X*X
130 NEXT X
```

こうすることでプログラムは分かりやすいし，計算が少ない分だけ処理が速い．（行番号110でXの終端を1.01としたのは，あらかじめ誤差が生じたときの対応である．32節参照）

〈例3〉　二つの数値i，jを3で割った値の組（i/3，j/3）をグラフィック座標として，その座標上に点を打とう．

神経質な人は次のようにコーディングするだろう．

```
300 X=INT(I/3):Y=INT(J/3)
310 PSET(X, Y)
```

すなわち，マニュアルにはグラフィック座標(X，Y)のX，Yは整数と書かれているから，上記のようにINT関数を用いたのである．しかし，これは次のようにコーディングすべきである．

```
300 PSET(I/3, J/3)
```

この理由は，必要な切り捨てはBASICがしてくれる，ということである．

（注：　市販されている一部の機種では切り捨てではなく四捨五入される．）

BASICに頼れるところは，すべてBASICに頼ろう．

29 数式はそのままの形でコーディングするな

人間でもそうだが，コンピュータにおいて計算回数が少なければ，当然計算スピードは速くなる．

$$y=x^3+2x^2+3x+4$$

をコーディングするとき

```
100 Y=X * X * X+2 * X * X+3 * X+4
```

とすると演算の数は合計8個であるのに対して，

```
100 Y=((X+2) * X+3) * X+4
```

とすると，演算の数は合計5個になり，当然計算スピードは上がるはずである．このように数式に少しの工夫を加えることで処理効率が向上することが多い．

〈例〉 関数 $y=x^2$ において区間 $a \leqq x \leqq b$ で x 軸とこのグラフとで囲まれた面積を求めてみよう．

これを一番簡単な区分求積法で求めてみる．数学の有名な公式を用いる．

$$S_n=\sum_{k=1}^{n}(x_k)^2\Delta x \ (\Delta x=\frac{b-a}{N},\ x_k=a+k\Delta x)$$

ここで N を十分大きくとると S_n は面積の良い近似値となる．この数学の式をそのままコーディングすると次のようになるだろう．

```
270 S=0:D=(B-A)/N
280 FOR K=1 TO N
290   S=S+(A+K*D)^2*D
300 NEXT K
310 PRINT S
```

しかし，少し考えれば次のようにした方が良いことが分かる．

```
270 S=0:D=(B-A)/N
280 FOR K=1 TO N
290   S=S+(A+K*D)^2
300 NEXT K
310 S=S*D
320 PRINT S
```

すなわち，数学上の有名な公式を次のように変形しておくのである．

$$S_N=\Delta x\left(\sum_{k=1}^{N} f(x_k)\right)$$

こうすることで，ずいぶん積*の数が減少する．

公式などを，そのままの形でコーディングしてはならない．

Memo　ループ計算と数式変形

コーディングにおいて，処理時間の短縮をねらうとき一番注意するのがループ計算内での命令記述である．そこではプログラム中，しばしば実行されるところなので簡潔な命令を書いておかねばならない．そして本節の注意も，その中で生きてくるのである．（何も1回しか実行されないところで，本節の技法は無意味である．）

そのループ計算内では次の二つを念頭においてコーディングすべきである．

(1) もっと簡単で短いBASIC命令はないか．

(2) もっと演算回数が少ないコーディングのしかたはないか．

30 数学を武器とせよ！

高校数学程度の教養を身につけていると，それがプログラムの処理効率を向上させるのに大きな武器となる．

〈例1〉　二つの変数X，Y（両方とも正数とする）の各々の常用対数の和を計算し，それを変数Sに代入せよ．

この例を単純にプログラミングすると次のようになる．

```
300  S=(LOG(X)+LOG(Y))/LOG(10)
```

しかし，これではLOGというBASICに備わった数値関数を3度呼び出すことになり，計算速度は遅くなる．したがって，よほどプログラムが分かりにくくなることがない限り，我々は次のようにコーディングすべきである．

```
300  S=LOG(X * Y)/LOG(10)
```

これで計算速度の遅い数値関数を呼び出すのが2度に減った．FOR～NEXT文でこの計算を多くさせると，二つのプログラミングの差が非常に大きいことが分かるであろう．(もちろん，そのときにはLOG(10)も数値にしておくべきである．)

〈例2〉　変数Xの正弦値と余弦値とを掛け合わせ，それを変数Mに代入せよ．

これも〈例1〉と同じ理由で次の二つの文のどちらをとるべきかは自明であろう．

```
370  M=SIN(X) * COS(X)
370  M=.5 * SIN(2 * X)
```

ここで数学上の有名な公式（倍角の公式）

$$\sin 2x = 2 \sin x \cos x$$

を用いている．この例1，例2の技法を用いると次のような数値計算で，計算速度はおおよそ半減する．

〈例3〉　$\int_{\pi/4}^{\pi/3} \{\log(\sin x) + \log(\cos x)\}\, dx$ の値を区分求積法を用いて求めよ．

```
100 N=100 :P=3.14159
110 A=P/4:B=P/3:D=(B-A)/N
120 S=0
130 FOR X=A TO B STEP D
140   S=S+LOG(SIN(X)*COS(X))
150 NEXT X
160 S=S*D
170 PRINT S
180 END
```

行番号140が示しているように，単に与えられた式をプログラムの式とすべきでなく，いかにして計算速度が上がるかの工夫をすべきである．

最後に，数学の公式を用いて処理時間を短くするときの注意点を述べよう．数学を武器として式変形をしたものをコーディングすると，どうしてもプログラムが分かりにくくなってしまう．処理効率を上げてもプログラムが難解となり，万一バグが発生してもなかなか対応がとれなくなっては元も子もない．したがって本節の技法を用いるときには，このことに十分気をつけねばならない．次の例は，このことを考えたコーディング例である．

```
370 M=.5 * SIN(2 * X) : ' M=sin(x)cos(x)
```

これは〈例2〉のコーディングを改良したものであるが，このコメント文の主旨については説明を要しないことと思う．

数学を用いて計算式を簡単にする努力をしよう．

Memo　効率か，わかりやすさか？

本節では，効率のよいコーディングはわかりにくくなるという印象を与えてしまうが，経験上次のことがいえる．すなわち，多くの場合，わかりやすいプログラムは簡潔であり，簡潔なプログラムは効率がよい．

31 使用ずみ資源はすぐに返却

ちょっとした設計の工夫や，コーディングのしかたによって，我々は限られたパソコンシステムの資源を有効に利用できるようになる．そして，この工夫を怠ることで，本来購入の必要のないRAMやディスク装置を買わざるをえなくなることがある．この節ではこの点について，コーディング時における心配りを述べよう．

一つの例として，ディスクモードのシステムの立ち上がり時にシステムは次のような入力要求を我々にしてくる．

How many files ?

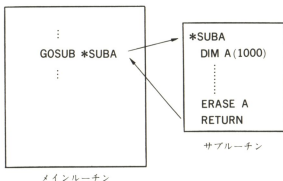
我々はこの入力要求に対して，同時に用いるディスクファイルの数を入力するわけであるが，資源の有効利用という観点から，この数はできるだけ小さい方がよい．すなわち，大きな数値を入れるとそれに比例してシステムがワークエリアおよびインタフェースエリアとして我々ユーザが使用できるメモリ内に領域を確保してしまう．必然的に利用できるメモリが小さくなってしまうのである．

また同時にアクセスするファイルの数を大きくとり過ぎるとディスク装置そのものを新たに購入しなければならなくなってしまう．

このようにディスクファイルについては同時に使用する数を可能な限りできるだけ小さくすることが望ましい．そして，このことを念頭に置いてプログラム設計をすべきである．またコーディングに際しては既に使用ずみとなったファイルはすぐにCLOSEすべきであるということになる．すなわち開いたファイルをいつまでも閉じずに放置しておくことは大変不謹慎なのである．開いたファイルはできるだけすぐにまとめて使用し，そして使い終ったら即座にCLOSEすべきなのである．

もう一つの例として配列を考えてみよう．我々は配列を安易に用いたために次のようなエラーメッセージをBASICに出させてしまうことがよくある．

out of memory

これはプログラムの設計において我々がメモリの見積りに失敗したことが原因であるが，このメッセージが出力されたからといってあわてて設計変更したり，新たに増設RAMを購入したりすべきではない．まずコーディング段階でしっかりした資源の有効利用をしているかをチェックすべきである．たとえばモジュール化されたプログラムで，一つのモジュール内でしか使用していない配列は，他のモジュールに実行を渡したときにきちんとERASEされているかとか，ある一箇所でしか使用していないワーク用の配列がいつまでも生き残っているのではないか，といった初歩的なチェックが意外に問題を解決してくれるのである．

以上，ディスクファイルと配列の二つの例で資源の有効利用について触れ，またそのコーディング上の対策として使用ずみ資源（ファイル，メモリなど）の即時返却を述べた．この二つの例からわかるように，コンピュータの限られた資源は常にフルに利用するという心配りが大切である．使ったままプログラムが終了するまでそれを放置しておくのはもったいない話である．

使用ずみファイルはすぐにCLOSE，利用ずみ配列はすぐにERASE．

32 等号の判定は慎重に

次の例をプログラミングしてみよう．

〈例〉 0.1＋0.2＋0.3＋……＋1.0 の和を求めよ

これに対して次のようなコーディングをしたとする．

```
100 S=0:X=0
110 '
120 IF X=1 THEN 160
130 X=X+.1
140 S=S+X
150 GOTO 120
160 PRINT S
170 END
```

実際，これをRUNするとパソコンは沈黙する．すなわち行番号20〜60の間をぐるぐる回る計算を延々と続けているのである．こういう状態を“無限ループに陥る”といって，我々がしばしば経験することである．

上の例の無限ループの原因は行番号120にある．21節でも述べたようにコンピュータは数値を2進表示するが，このとき0.1を2進数で表わすと

0.00011001……

という無限に続く数となる．コンピュータはこれを有限桁で切ってしまうので，結局0.1という10進数を2進数で正確に表現しなかったことになるのである．このため，行番号40は我々の理論通りに等号を満すことがないため，絶対に行番号160に脱出できないのである．

我々はプログラムのコーディングにおいて，IF文の中に数値変数の関係する等号をおくことに大変な注意を払わねばいけない．理論的には等号が満されても，パソコンの表現方法ではそれを満していないことがあるのである．

このようなことを回避するために，我々は次のようにコーディングすることを勧める．すなわち，等号判定はそれよりも条件のゆるい不等号の判定に書き変えよということを原則にするのである．この節の例では次のようにする．

```
120 IF X>=1 THEN 160
```

こうすることで条件がゆるくなり，万一パソコンの内部表現に誤差が生じても大丈夫になるのである．

等号の判定文はなるべく避け，不等号に書き変えよう．

Memo FOR～NEXTで無限ループ回避

この節の例はFOR～NEXTにもいえることである．

```
100 S=0
110 FOR X=0 TO 1 STEP .1
120   S=S+X
130 NEXT X
140 PRINT S
150 END
```

これは本節のプログラムをFOR～NEXTを用いて書き換えたものである．実行してみると

4.5

という値を表示する（正答は5.5)．すなわち，GOTO文をなくしてFOR～NEXT文にしても，無限ループは回避できても正解は得られないのである．一部にFOR～NEXTを用いて無限ループ対策にできるということがいわれているが，確かにそれは回避できてもプログラムには誤りが生じる(58節を参照のこと)．

33 入力データのチェックは厳密に

キーボードから与えられたデータについては，我々はそれを厳密にチェックせねばならない．キー操作をする人が誤ることを常に仮定してプログラムをつくるのである．そして，もし，キー操作ミスでプログラムが誤動作をしたなら，その責任はオペレータではなくプログラム作成者にある．

例として次のプログラムを考えよう．

```
100 'root calculation
110 INPUT "x=";X
120 R=SQR(X)
130 PRINT "root x =";R
140 END
```

これは入力されたXの値に対して，その平方根を求めるプログラムである．「何，簡単だ！」と思われるかも知れないが，きちんとしたプログラムに仕上げようとすると結構大変である．

上記のプログラムはまだ未完成である．もし，Xに負の値が入ったとすると行番号120でプログラムは異常終了してしまう．そこで行番号110と120との間に次のようなチェックの命令が必要となる．

```
115 IF X<0 THEN 110
```

これで完成か，というとまだである．もしキー操作を誤って行番号110の入力要求に“A”という文字を入れてしまったとする．すると次のようなメッセージが出力されてパソコンは再び入力待ちとなる．

```
Redo from start
```

プログラム・ユーザはこのメッセージを見て，「何だ，この意味は？」と困惑してしまう．そこで上記のプログラムは大幅に変更されねばならない．すなわち文字変数を用いて入力要求を出すよう改良するのである．

```
100 'root calculation
110 INPUT "x=";X$
120 X=VAL(X$)
```

```
130 IF X$<>"0" AND X=0 THEN 100
140 IF X<0 THEN 100
150 R=SQR(X)
160 PRINT "root x =";R
170 END
```

最初に比べてずいぶんと複雑になってしまった．しかし，これでも不十分なのである．もし入力要求に対して次の値を入れたとしよう．

1E+60

すると行番号20で“overflow”というメッセージが出され，異常終了してしまう．

このオーバーフロー対策は容易ではない．きちんとした文字処理をしてチェックしなければならないのである．このように，たかだか平方根を求めるプログラムを作成するだけでも，入力データが絡むと大変になるのである．このことをしっかり頭に置いておかないと思わぬところでバグを発生させ困ることになってしまう．

入力要求命令の後には必ずデータチェックを厳しく．

Memo ファイルセーフ（Fail safe）

システムが高度になればなるほど，操作ミスに対する対策を万全にしなければならない．原子炉で作業員がちょっとした操作ミスをしたくらいでそれが爆発するようでは危険きわまりないのである．このようにミスに対して万全な策をとろうということを表現したものに表題の言葉がある．すなわち，まったくの無知な者がそのシステムを触っても異常を起こさないようにせよ，ということを表現しているのである．

第3章

使いやすいプログラムへの心くばり

パソコンのプログラムは，その作成者だけが使用することにとどまらず，色々な人々の利用に供されるようになっている．このような現実の中で，我々の作成するプログラムは，どのように設計ないしはコーディングされなければならないか，すなわち，利用する人間にとって使いやすいようにするにはどうすべきかをこの章で述べよう．

34 パソコンを人に近づけよう

パソコンは，OA 機器として広く社会にゆき渡ったが，よく次のようなことがきかれる．「あじけない」とか「使いにくい」と．今まで女性事務員などがやっていた計算をコンピュータに置き換えたのだからある程度はしかたのないことだが，それらの批判には我々プログラム作成者にも責任がある．すなわち，汎用性をねらったり安易性を求めるあまり，プログラムを使う側の立場に立つことを忘れていたのではないだろうかという反省である．

たとえば，一つの処理が終了したとき，画面に次の二つのメッセージが出力されたとすると，どちらが人々に和らぎを与えるだろうか？

END

御苦労様でした

もちろんケースバイケースでどちらが良いともいえないが，明らかに右の方がユーザに安らぎを与えるであろう．

また入力要求をするときにも，単に INPUT 命令だけを用いたのでは画面に“？”マークが出力されるだけでデータを要求しているという気分を起こさせない．そこで入力要求文（**プロンプト文**）をブリンク（点滅）させることで入力要求を？マークよりさらに強く要求する，といった配慮も大切である（45 節参照）．

名前＝？

さらにテキストを色分けし，入力要求文は赤，使い方の説明は青，プログラムの出す警告等は黄色などとすると一層パソコンが人に近づく（ただし，色は多用し過ぎるとかえって見にくくなる）．

以上は入力要求の出し方を例に上げたが，画面全体の配置についてもデータ量が多いときにはできるだけテーブル化するという心配りが大切である（もちろん常にという訳ではないが）．次の入力例を見てみよう．

```
名前＝？　　ヤマダタロウ
生年月日　　年＝？　32
　　　　　　月＝？　10
　　　　　　日＝？　14
本籍コード　＝04
```

名　　前		ヤマダタロウ
生年月日	年	32
	月	10
	日	14
本籍コード		04

どちらが使いやすいかは一目瞭然であろう．

プログラムが発するメッセージについても同様である．たとえば入力エラーがあったときにただ単に“エラー”と表示するよりも，そのメッセージとともにブザーを鳴らすという心配りが大切である．特に重大なエラーや緊急に入力を要求するときにブザーを細かく鳴らすと，プログラムの利用者はどんな画面上のメッセージよりもはるかに強くその緊急性を解するものである．

このように，多少の努力を払うことでパソコンは非常に生き生きとした道具に変身してくれる．我々はできうる限りパソコンを人間的なものにしたてるよう努力せねばならないであろう．

ちょっとした配慮を用いてパソコンを温かみのあるものにしよう．

35 BASICにエラーメッセージを出させるな

コンピュータのことをよく知らない人が，オペレータとしてパソコンの前に座り，我々の作成したプログラムを利用して計算処理をしていると仮定しよう．

たとえば，次のような入力要求をプログラムが出したとする．

従業員番号＝？

この入力要求に対して，オペレータが555と入れるべきところを誤って55Aと入力してしまったとする．このときディスプレイの画面上に次のようなメッセージが出力されたとすると，そのBASICを何も知らないオペレータは困惑してしまうであろう．

```
従業員番号=?  55A
?  Redo  from  start
?
```

またプログラム中にバグがあり，計算処理の途中で

```
divided  by  0
```

というメッセージが表示され，プログラムが異常終了したとする．何も知らないオペレータはどうしてよいものかと戸惑うばかりである．

これらの例のように，プログラムを実行中BASICがエラーや警告のメッセージを出力することがあるが，我々は極力このようなことがないよう注意せねばならない．我々が作成したプログラムを我々自身が使うならともかく，他の多くの人が使うことを想定するなら，BASICの出力するメッセージは決して分かり良いものではない．

第一の例の入力要求は次のようになされている．

```
INPUT  "従業員番号=";  NO
```

NOは従業員番号が入る数値変数である．我々はこの数値変数を文字変数にしておかねばいけなかったのである．すなわち次のようにコーディングすべきであった．

```
100 INPUT "従業員番号="; NO$
110 NO=VAL(NO$)
120 IF NO=0 THEN GOTO *ERRSUB
```

ここでラベル*ERRSUBは入力エラーに対処する処理の先頭ラベルとする．このように入力変数を文字変数にすることで，誤ったキー操作をしてもほとんどの場合我々のプログラム内で責任がとれ，BASICがメッセージを出すことを回避できる．

第2の例のようにプログラム内にバグがあり，そのためにBASICがエラーメッセージを出すことがあるが，これを回避するには次のようにすればよい．プログラムの先頭に次の1文を入れておくのである．

```
ON ERROR GOTO *ERRSUB
```

*ERRSUBというラベルはエラーに対処するための処理の先頭ラベルであるとする．この一文をプログラムの先頭に入れることで万一バグなどの理由でプログラムに異常が生じても，BASICに頼ることなく我々自らの責任でそのエラーに対応できるのである．また，このエラー処理ルーチンに次のようなメッセージを出力する用意をしておけば親切である．

```
1000 *ERRSUB
1010 PRINT "専門家をお呼び下さい！"
 ：    ：
```

このようなメッセージを出すことでプログラムの利用者は安心して対応がとれ，またエラーの現状が保存されているので我々プログラム作成者はすぐにデバッグが可能である．

BASICにメッセージを出させず，自分で責任をとろう．

36 入力ミスは日常茶飯事

我々がプログラムを使うときに，一番疲れるのがキー操作であろう．この疲れる原因の一つに“キー操作を誤ってはいけない”という緊張感がある．プログラム設計者は，したがって入力においてこの緊張感から，プログラム使用者（オペレータ）を解放せねばならない，そのためには“間違っても大丈夫”という安心感を与えるようなプログラムでなければならない．ちょっとしたキーインミスでプログラムが異常動作をするようでは困り物である．

ここでは入力のためのINPUT命令の使い方をいくつか列挙してみよう．

(1) INPUT命令の右辺にある入力変数は文字型にすること

これについては35節で既に述べた．すなわち

```
INPUT A
```

という命令に対して我々がキーインミスをし，文字などの数値以外のキーを押したとすると，BASICが「Redo from start」などという警告のメッセージを出してしまう．それではプログラムユーザが困惑してしまう．

(2) INPUT命令の右辺に複数の入力変数は置かない

これは，次のような命令の記述はやめよう，ということである．

```
INPUT A$, B$, C$,
```

このような使い方をすると，入力変数と入力個数が不一致のときBASICは「Extra ignored」といった警告メッセージを出力する．それでは使いにくい．

(3) 入力桁数のチェックを

我々はプログラムで扱うデータの桁数を予想してそのプログラムの設計をするが，入力されるデータについてはその桁数はまったく予想できるものではない．たとえば，いま考えている計算処理では単精度計算で十分であると予想しプログラミングしても，INPUT命令で入力させる数値の桁数はオペレータのキーインミスによって予想不能である．したがって，このような場合，INPUT命令は次のように用いるべきである．

```
      :
300 INPUT A$
310 IF LEN(A$)>6 THEN PRINT "入力ミス" : GOTO 300
320 A=VAL(A$)
      :
```

LEN (A$) とは文字変数 A$に格納された文字データの長さ（文字数）を値としてもつ関数である．単精度では通常6桁以下の有効桁となるから，7桁以上の数値を入力されても正確なデータ処理ができない．行番号310はそれをチェックしている．このように，どんな長さのデータを入力してもプログラムが正常に動くように入力段階のチェックを厳しくしておかねばならない．

(4) 入力データの条件をしっかり把握

次のようなコーディング例を考えてみよう．

```
      :
500 INPUT K
510 ON K GOSUB *KEISAN, *INSATU
      :
```

すなわち，入力された整数Kに対して，それが1なら計算ルーチン，2ならば印刷ルーチンに制御を渡す論理とする．このとき，Kは明らかに1，2以外の数値をとらないと予想している．したがってこれは次のようにコーディングされるべきである．

```
  :     :
500 INPUT K$ : K=VAL(K$)
510 IF K<1 OR K >2 THEN PRINT "入力ミス" : GOTO 500
520 IF K< >INT (K) THEN PRINT "入力ミス" : GOTO 500
  :     :
```

ここで行番号520は整数か小数かのチェックをしている．

この例のように，入力されるべきデータの性質についてチェックできるところはこと細かにチェックすべきである．

入力されたデータは徹底的に調べよう．

37 メッセージは日本語で

コンピュータは多くのエレクトロニクス製品と同様発祥地が欧米であるため，その用語はほとんどが英語である．そのために英語を多用することがコンピュータの本質であるかのように，マニュアルやディスプレイ画面に表示する文章を英語で記述する人がいる．

パソコンは日本社会のあらゆるところで活用されている．そして優れたプログラムはあらゆる知識層の人々に使用されることになる．このとき，英語で書かれたコンピュータメッセージをすらすら読めない人々も多くいる，ということにも気をつけねばならない．

日本で市販されているパソコンのほとんどの機種には最低カタカナを使える機能がついている．我々は英語のキーばかりに頼らず，この標準的に備えられているカタカナのキーを利用すべきである．たとえば「リターン・キーを押して下さい」という入力要求のメッセージ（プロンプト文）を出力するとき，次のようにコーディングすべきではない．

```
270  INPUT  "Type in RETURN key" ; D$
```

せっかくカナキーがあるのだから，次のように記述すべきである．

```
270  INPUT  "RETURN キーヲオシテクダサイ"; D$
```

もし漢字 ROM が備わっているならばまさに原文通り次のようにすべきである．

```
270  INPUT  "RETURN キーを押して下さい"; D$
```

パソコンは計算処理の道具である．その道具に利用者が近づく努力をするのではなく，利用者にその道具が近づくようにすべきである．我々はプログラムを作成するとき，この点に十分注意せねばならない．

メッセージは分かりやすい日本語で表示しよう．

38 いつでも中断できるように

科学計算やファイル処理を伴う計算処理などはどうしても時間を費やすものである．その計算途上急用のため中断せねばならないことが実際の運用上よくある．そこで我々は長い処理時間を要するプログラムにおいては処理途中でいつでも中断できる機能を用意せねばならない．

計算途上の中断を可能にするとは次の二つの意味に解釈される．

(1) 再度計算させようとするとき，その中断の時点から出発できる．

(2) 再度計算させようとするとき，新たに計算前の状態から出発する．

この(1)または(2)の意味の中断を完全にサポートするのは非常に大変な場合が多い．その実現については各計算処理についてケースバイケースで検討するしかないが，この節ではいかにプログラムユーザにその中断機能を提供するのかを示そう．

通常我々は次の BASIC 命令を用いてユーザに中断機能を提供する．

```
  :
140  ON  STOP  GOSUB  2000
  :
280  STOP  ON
  :
2000  ' stop process routine
  :
  :
2650  RETURN
```

すなわち，STOP キーを押すことでユーザは処理を中断できるようにするのである．我々プログラム作成者はストップキー割り込みで飛んだ先のサブルーチン（ストップ処理ルーチン）で中断処理（ユーザがここで処理を中断してファイル等が破壊されないような処理など）をし，制御をメインルーチンに戻す．このような論理で中断機能を提供することで，プログラムは非常に使いやすいものになるのである．

急用のとき席をはずせるような構造をプログラムに与えよう．

39 マニュアルを読ませず画面で語れ

人に利用されるプログラムを作ったときには，その使い方を示す解説（マニュアル）が必ず用意されねばならない（9節参照）．しかしプログラムの利用者にとって，そのマニュアルを読みながらプログラムを運用して行くのは本当に苦痛である．いくら親切なマニュアルを作っても，ディスプレイとマニュアルとを見ながら作業をするのは人間にとって不自然なのである．

我々はプログラムを作成する際，できるだけ画面を見るだけで作業が進められるように丁寧な解説をディスプレイに表示すべきである．たとえば次の左右二つの画面を見てみよう．

```
(1)  CALCULATION
(2)  PRINT OUT
(3)  DATA—INPUT

     WHICH?
```

```
(1)  計算をする
(2)  印刷をする
(3)  データを入力する

(1)から(3)の番号を入力して下さい？
```

どちらの方が使いやすいかは一目瞭然であろう．いくらしっかりしたマニュアルが完備していても左の画面のようなプログラムを作ってはいけないのである．画面が語る部分は大なのである．

親切な画面をつくるときには次の BASIC 命令を用いると便利である．

ON　HELP　GOSUB　文番号

これは以後 HELP　ON という命令を実行すると，HELP キーが押されたなら GOSUB の次の文番号に分岐させる命令である．我々はこの文番号から始まるサブルーチンに操作手順を示す画面を作成する処理を用意しておくのである．こうすることでプログラムユーザが作業手順が不明になったとき，単に HELP

キーを押すだけで次の作業のしかたが分かるのである．いちいちマニュアルを調べるよりもはるかに効率がよいことは明らかであろう．

```
    :
1400  ON  HELP  GOSUB  5000
    :
1870  STOP  ON
    :
5000  ' HELP キー  ニヨル  ワリコミ
    :
    : 画面に作業手順を解説する処理
    :
5550  RETURN
```

画面を見ただけで作業手順が分かるようなプログラムを作ろう．

Memo **BASIC で使える特殊キー**

38節の STOP キーや本節の HELP キーのように，特別なキーとして定義できるキーは他に FUNCTION キーだけである．しかし，これらのキーを上手に使うことで，プログラムは非常に使いやすいものに変容してゆく．本節の HELP キーの使い方も良い例であろう．

40 プリンタの紙詰まり対策を

パソコンのシステムは，非常に信頼性の高いものである．ハードウェアの障害などはあまり気にしないですむ．その信頼できるシステムの中にあって唯一信頼できないものがプリンタである．我々はしばしばプリンタに出力中，紙詰まりに出会う．これはコンピュータシステムの中で一番物理的な動きをするのがプリンタであるからである．したがって我々はプログラムを作成するとき，プリンタの紙詰まり対策を考慮しておかなければならない．

この紙詰まり対策は大変むずかしい．紙詰まりを起こしたすぐ直後にプリンタがそのことをCPUに知らせてくれればよいのだが現実にはそうはなっていない．したがって多くの場合紙詰まりが発生したとき再び最初から印刷し直さねばならないことになる．

紙詰まり対策としては次の二つの対策が考えられるだろう．

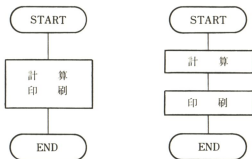
(1) 正常に打ち出したところを指定し，その次から出力できるようにプログラムを設計する．

(2) 出力内容をディスクにしまっておき，何度でもそこからはき出せるようにしておく．

(1)を実現するのは(2)よりも大変である．しかし(1)を可能にするには(2)を前提とする場合が多い．したがって(2)は必ず守っておかねばならない対策であろう．

(2)の意味は，計算をしながら印刷するのではなく，計算結果はすべてディスクに保存し，計算が終了したらおもむろに印刷を始めよ，ということである．すなわち次のようなフローチャートにおいては右の論理を採用せよということである．

こうすることで，万一紙詰まりが発生しても計算処理は飛ばして印刷だけを繰り返すだけですむというメリットが生じるのである．

(2)の対策がとられていれば，(1)の対策を実行することは容易である．ディスクにためられた処理結果に番号をつけておき，その番号をキーボードから入力することによって指定された処理結果をプリンタに印刷させることができるの

である．

(1)の対策がとられることは理想だが，我々は最低(2)の対策を念頭においてプログラムを設計すべきである．プリンタへの出力はコンピュータ業務の中で一番時間をくうところであるが，その際，一時もプリンタから離れずに紙詰まりが発生しないよう監視していなければならないような設計をすべきではない．

紙詰まりが起きても利用者が動揺しないような対策をとろう．

> **Memo**　**パソコンのプリンタ**
>
> パソコンのプリンタは安価なものが多く紙詰まりは不可避的である．(汎用コンピュータ用のラインプリンタもよく紙詰まりを起こすが！）したがって我々の対策としてあまり長いひとまとまりのデータをプリンタに出力させないことである．全部を出力しなければ意味をもたないようなリスト出力はできるだけ短くせよ，ということである．このような自衛手段をとらないとストレスがたまってしまうことが多い．

41 キー操作はできるだけ少なく

コンピュータ業務で一番苦労するのがキー操作である．また誤りが一番多く発生するのも，この操作に関係するところである．したがって，このキー操作をいかに容易にするかが優れたプログラムの一つの条件である．

キー操作を簡単にする方法として一番有効な方法は，キーインの回数を減らすことである．すなわちキーをたたく回数が減らされればキー操作の時間は短縮され，またその分誤りが少なくなるはずである．

キー操作の簡略化に一番重要なのは，設計段階における入力データの整理である．どのデータが不可欠で，どのデータが入力不要かを，しっかりまとめた上でプログラミングすることが基本となるのである．たとえばある会社の男女の従業員と総社員数を入力するのに

```
100  INPUT  "man number=" ; MN
110  INPUT  "woman number=" ; WN
120  INPUT  "total number=" ; TN
```

とコーディングしてはいけないのである．当然行番号 120 は次のようにすべきである．

```
120  TN=MN+WN
```

（もちろん，チェックのために上記のようなコーディングをする場合があるが．）こうすることで一回入力回数が減る．これはおおげさな例であるが，主旨は理解されるであろう．

キー操作において次のような工夫も大切である．入力する文字が1文字と決まっている場合，次のようにコーディングするのは不親切である．

```
270  INPUT  "ALPHABET=" ; C$
```

これは一つの英文字を要求する命令とする．このとき次のようなキー操作となる．

〈例〉 [A] [RETURN]

すなわち二つのキーを押すことになる．このようなときには次のようなコーデ

ィングを勧める.

```
270 PRINT "ALPHABET=?"
280 C$=INKEY$ : IF C$="" THEN 280
```

こうすることでキー操作は[A]というキーを押すだけですむようになる. たとえば小学校の成績(5段階)を入力するプログラムなどではこの技法を用いることで数千回の打鍵が省略できるのである.

入力の文字数が一般的にN個と決まっている場合(N≧2), 上の方法の拡張として次の命令を用いることを勧める.

```
INPUT$(N)
```

たとえば6桁の従業員番号を入力するとき

```
270 INPUT "従業員番号="; NO
```

とするのではなく, 次のようにすることを勧めるのである.

```
270 PRINT "従業員番号=";
280 N$=INPUT$(6)
290 NO=VAL(N$)
```

こうすることでINKEY$と同様, RETURNの打鍵が省けるのである.

以上の例のように, ちょっとした心配りで入力回数はずいぶん減り, 入力操作が簡単になることを銘記してもらいたい.

キー操作が少なくなるようプログラムが面倒を見よう.

42 RUN命令も知らない人への配慮を

パソコンは，昨今電卓並みに利用されつつある．社会や学校，研究機関のすみずみに行き渡っている．このような中でパソコンを利用する人にパソコンの利用方法についての基礎知識を期待することは不可能になってきた．かつては「マニュアルを読め」ですんだことが今では通用しなくなっている．

大型コンピュータには専門のオペレータがついている．しかしパソコンにはそのような人がいるはずもない．電源を入れ，ディスクをセットするだけでもやっとという人を対象にプログラムを作ることが大切である．そのためにはエラー対策をしっかりし，画面へのメッセージを丁寧にするということが必要であり，誰にでもわかるマニュアルを完備することも重要である．これらのことについては他の節に譲ることとして，ここではLOAD, RUNというコマンドも知らない人への配慮として**オート・スタート機能**について触れよう．

このオート・スタート機能とは，CPU本体に電源を入れると既にセットされているディスク上のプログラムをCPUが自動的に読みにいく機能のことである．そして読み終わったプログラムは自動的にRUNされるようになっている．このオート・スタートは，ディスク上にある管理テーブルの一部分を書き換えることで実現できるが，パソコンの多くにはそれがユーティリティープログラムとして備えられている．その動かし方についてはマニュアル類を参照してもらいたい．

このオート・スタート機能で代表されるように，我々はできる限り，パソコンについての操作からユーザが解放されるように準備をしておくべきである．そしてまったくのパソコン音痴の人も，パソコンが利用できるよう配慮しておくべきである．

システム操作(LOAD，RUNなど)から，利用者が解放されるよう心配りをしよう．

43 黙って待つ時間はせいぜい10秒

コンピュータの計算処理の速度は，我々に比べて非常に速い．しかし，いかにコンピュータでもあらゆる処理を瞬時に終わらせることはできない．特にファイル処理やソーティング処理では，我々はよく待たされる．コンピュータの計算途中ではコンピュータは黙っている．したがって，プログラムの構造を知らずただわけも分からずに待たされているプログラム・ユーザは不安になり，またイライラしてしまう．

我々がプログラミングする際，常に処理効率というものを考えるべきである（第6章参照）．特にBASICにおいてはその言語の特性上，ちょっとした工夫が多大なスピードアップにつながるのである．また，プログラミング上有名な計算技法（ソーティングなど．90，91節参照）をしっかりマスターしておくべきである．そうすることによって待ち時間はかなり短縮される．

我々は，ここでどうしても待たせることを避けられないときに配慮すべきプログラマのエチケットを述べよう．すなわち10秒以上の計算処理またはファイル処理にプログラムが突入するとき，プログラマはあらかじめディスプレイにその旨を表示するのである．プログラマは時間のかかる場所をあらかじめ予想することが可能であるから，これくらいの心配りは容易である．

このようなメッセージが出されると，「ではお茶でも飲んでこようかな」というように利用者は安心するものである．コンピュータだけが必死に計算をして，利用者はわけも分からずにただ待たされている状態は絶対に回避すべきである．

```
コンピュータが計算中です．
あと５分程お待ち下さい．
```

待たせるときにはその理由を示すか，待ち時間の予想を示すべきである．

44 チェックリストの準備を

たくさんのデータ入力を伴うパソコン業務において，常に心配なのが入力ミスである．したがって，プログラムは二重三重にデータをチェックすべきであり（33節参照），またいつでもデータ訂正ができるようにすべきである（8節参照）．しかし，このようなチェック・訂正を行っても，最後のよりどころとなるのが，プリンタで出力されたファイルの中味のコピー（**ハードコピー**）である．画面上で何回チェックしても我々の不安は消えない．しかし，プリンタから出力された入力データのリストを，元本とつき合わせれば，非常に安心できるのである．データ入力のルーチンには，必ずそれと並行して入力データの一覧表を作成するルーチンを並設しておかねばならないのである．

この一覧表作成ルーチンで出力されるデータの形式は簡素でよい．ダンプリストと同じ発想で作られるものであるから，なるべく出力スピードが速くなるように行間をつめて，すき間をできるだけ少なくするようにする．次のリストはその形式の一例である．

社員番号	総支給額	税　金	社会保険	組合費	男　女	職　種
601655	417,247	24,744	15,743	4,700	1	A2
644111	376,697	29,713	14,518	4,500	1	A4
654330	355,473	18,857	14,315	4,500	1	A4
654115	367,865	20,013	14,440	4,500	1	A4
734196	285,618	15,117	12,715	4,100	0	B1
756773	271,093	14,883	12,505	4,100	1	A5
784350	195,036	10,117	8,307	3,500	1	A7
804464	222,769	12,315	10,318	3,700	1	C2

当然このような形は実際に人に渡したりする出力形式と異なる．給料明細，住所案内，等で人々に渡すリストは各データについてもっと説明を丁寧にしなければならない．ここでいうチェックリストとはデータ入力者が入力した内容

をその場で知りたいときに，できうる限り整理された形で迅速に出力されるファイル内容のリストのことである．

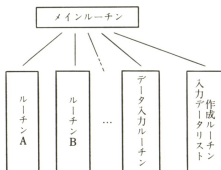
以下にこのチェックリストの要件をまとめてみよう．

(1) 素速く出力できること

(2) 情報をコード化し整理すること．(たとえば男女の区別などは 0 と 1 ですむ)

(3) 不要な情報は出力しない

(4) 全体として見やすいこと

このリストを作成するためのモジュールは全体の中で次のような位置づけがなされるであろう．

データ入力者に，入力したデータの内容の一覧表がすぐとれるように配慮しよう．

> **Memo　チェックリストと秘密保護**
>
> 本節のような機能をプログラムに組み込むとき，それが誰にでも利用されるようでは困る．多くの場合，入力したデータは秘密にしておきたいものである．したがって，パスワード等 (95 節参照) を準備し，しっかりした機密保護をサポートせねばならない．

第4章

BASIC命令活用法

BASIC 命令の便利で有名な使い方をここで示そう．

BASIC は高級な言語であり，少し工夫することで色々と有用な使用方法が見い出される．またそれらの工夫でプログラムが生き生きしたものにもなってくるのである．本節ではプログラミング途上しばしば用いられる BASIC 命令の活用法を示そう．

45 文字の点滅で入力要求

オペレータに入力要求を出すとき，たとえば

社員番号＝？

という形で，ただ単に？マークだけで画面上に入力要求を出したのでは，オペレータはあまり入力しようという気にはなれないだろう．しかし，もしこの入力要求のメッセージが点滅したとすると，オペレータは強くそれに関心を奪われることになるだろう！

―社員番号＝？―

オペレータはこの点滅でパソコンが何かデータを求めていることをすぐに感じとることができる．この入力要求メッセージの点滅について NEC PC-9801 について説明しよう．

まず点滅させない通常の方法では次の命令が考えられる．

```
INPUT "社員番号＝"; NO        (点滅しない)
```

これに対して点滅させるには次のようにする（画面は白黒モードで使用されているとする）．

```
300 COLOR 2
310 PRINT "社員番号＝";
320 COLOR 0
330 INPUT NO
```

すなわち，白黒モードで COLOR 文を上記のように使うと，上手に画面上の文字を点滅（**ブリンク**）させることができる．

ただしこの 4 ステップのプログラムでは，入力終了後も入力要求のメッセージはチカチカと点滅し，どうも美観をそこねてしまう．そのために，入力後には通常点滅を中止させる．それが以下のプログラムである．

```
300 COLOR 2
310 LOCATE A, B : PRINT "社員番号＝";
320 COLOR 0
```

```
330 INPUT NO
340 LOCATE A, B : PRINT "社員番号"
```

すなわち，ブリンク解除，(COLOR 0) 後，再び入力要求のメッセージを上書きするのである．

画面上で入力要求のメッセージをブリンクさせれば入力要求の？マークは不要になる．また，ない方が美しい．？マークのとり方については48節を見てもらいたい．

パソコンは画面を通してオペレータと意志を通じ合うものである．したがって，できるだけその疎通を“人間的”に分かりやすくすべきである．この点滅（ブリンク）の工夫もその一つの例である．

文字の点滅などを用いてパソコンを人間に近づけよう．

Memo リバース表示

白黒の画面上で，ある特定の文字列を強調する方法として，ブリンクさせること以外に，文字のリバース（反転）という方法がある．これは文字の白とバックグランドの黒を逆転させる方法である．PC-8801/9801では これを次の指定で行うことができる．

```
COLOR 4
```

もし，もっと強調したいと思うときにはリバースしてブリンクさせれば その効果は大である．これも NEC PC-8801/9801 では次のようにサポートしている．

```
COLOR 6
```

46 文字の色で意味を区別

我々は，常にプログラム利用者が使いやすいよう心がけてプログラミングすべきである．画面を見たときに，すべて同一色の文字でテキストが記述されているとき(特に多くの行が並んでいるとき)，非常に見にくいことは明らかである．そこでテキストの色を区別し，色によって意味を違えるようにすることは，使いやすさを向上させるであろう．このやり方を NEC PC-8801/9801 で説明してみよう．

まず次の画面を見よう．

```
[数学の学習]
2次関数 y=ax^2+bx+c のグラフ
a, b, c の値を入力しなさい.
    a=? 0
    b=? 2
    c=? 3
a=c は条件に合いません. もう一度入力しなさい.
    a=?
```

これは CAI (コンピュータによる教育) の画面の一例だが，単色の画面では何か訴えるところが少ないであろう．そこで通常のメッセージは白で，入力要求は青で，そしてプログラムの発する警告は赤で表示することにしよう．上例の画面をこのようなカラー画面にするための BASIC 命令の流れを次に追ってみよう．

```
  :
1450  CONSOLE,,,1
  :
1690  CL1=7 : CL2=1 : CL3=2
1700  COLOR  CL1
```

```
1710  PRINT " [数学の学習]"
1720  PRINT "2次関数 y=ax^2+bx+c のグラフ"
1730  PRINT "a, b, c の値を入力しなさい"
1740  COLOR  CL2
1750  INPUT "a=" ; A
1760  INPUT "b=" ; B
1770  INPUT "c=" ; C
1780  IF A=0 THEN * REINP
  :
2500   * REINP
  :
2610  COLOR CL3
2620  PRINT "a=0 は条件に合いません．もう一度入力しなさい"
2730  COLOR CL2
2740  INPUT "a=" ; A
  :
```

行番号1450はテキスト画面をカラーモードにしている．行番号1700，1740，2610, 2730はこれ以降に画面に文字を書く際のその文字の色を指定している．

青…1

赤…2

白…7

実際にカラー画面を見るとその効果は大きいことが分かる．ぜひ多用してもらいたい．ただし，あまり色を多く用いすぎるとかえって画面が見にくくなってしまう．経験上3色位がちょうどよいと思われる．

豊富なカラー機能を用いて，画面を見やすくする努力をしよう．

47 暗証番号(パスワード)を隠すには

ファイル処理においては，その中味の秘密の保護は重要である．そこでよく用いられるのが暗証番号（パスワード）である．この暗証番号は人に見られては困るわけで，通常のINPUT命令で入力を要求すると画面にその番号が表示され不都合である．ここでは画面に入力文字を表示させない二つの方法を述べよう．(NEC PC-8801/9801に準じている．)

一つの方法は，INPUT$命令という BASIC 関数を用いることである．

```
     :
2700 PRINT "暗証を入力して下さい";
2710 CIPH$=INPUT$ (4)
2720 IF CIPH$<>ID$ THEN 2700
2730 PRINT "受け付けました"
     :
```

ここでID$には正しい使用者の暗証が入っているとする．また暗証番号は4桁とする．このコーディングに対応する画面は次のようになる．

```
暗証を入れて下さい
受け付けました
```

右図のように暗証は画面に出されない．

INPUT$はキーを押してもその文字は出力されないが，カーソルもまた移動しない．したがって入力したという気分をユーザに与えないのが難点である．これに対して，第二の方法としてこれから説明する方法は，カーソルが動くというメリットがある．この方法を示そう．

```
     :
1400 CONSOLE ,,,0
     :
2700 PRINT "暗証を入力して下さい";
2710 COLOR 1
2720 INPUT CIPH$
```

```
2730  IF CIPH$<>ID$ THEN 2700
2740  COLOR 5
2750  PRINT "受け付けました"
        :
```

このコーデングは前期のINPUT$を用いる方法と同じ画面になるが，ただ違うのは暗証番号を入れたときにカーソルが移動すること，および暗証番号を入力し終わったらRETURNキーを押すこと，である．NEC PC-8801/9801は白黒モードでカラー指定を行うとこのようなことを可能にさせてくれる．

パソコンが我々の日々の生活に密接に関与してくればくるほど秘密保護の対策は重要性を増してくる．この節で紹介した暗証番号の隠し方は，その対策の中で最も基本的なものの一つであるが，パソコンプログラムの設計には，ぜひともこのような配慮をしてゆくべきであろう．

パソコンにおいて，今後は秘密保護が重要となる．

Memo リバースシークレット

本節で示したコーディングでは入力文字がまったく隠されてしまうために暗証番号をしっかり入力したのかどうかがはっきりしないことがある．暗証番号を入力ずみであるということを画面に残すために，入力した位置に確かに入力した証拠を残したいことが多いのである．そのときこの表題の指定でINPUT命令を用いると入力した部分にバックグランドと同じ色のカーソル大の正方形が出力されていく．後で画面を見ると確かに入力したことがよく分かる．

このリバースシークレット機能は次のCOLORで指定する．

```
COLOR 5
```

48 入力要求の？を出さない方法

画面をしっかり枠組みしてデータの入力を欄の中に要求しようとするとき，INPUT 命令に伴う？マークが邪魔になることが多い．ここではこの？マークをとるにはどうするかを示そう．

このためによく用いられるのが次の命令である．

LINE INPUT

これは INPUT 文とほとんど同じように記述できるが，？マークは出力されない．下記の画面はその左に記したプログラムの入出力例である．

```
10 CLS
20 LINE INPUT "A=";A$
30 A=VAL(A$)
40 PRINT "A=";A
50 END
```

上例のように LINE INPUT の右辺に置かれる変数は文字変数に限られていることに注意しよう．

もう一つの方法として有名なのが，INKEY$関数を用いる方法である．たとえば入力が 1 文字に限られている場合には，次のように用いることができる（右の画面は入出力例である．）．

行番号 50 はキーから入力された文字を画面に表示するためのものである．

```
10 CLS
20 PRINT "A=";
30 A$=INKEY$
40 IF A$="" THEN 30
50 PRINT A$
60 END
```

```
A=3
OK
```

INKEY$関数はキーボードからの入力のみでBASICはその入力文字を画面には出力しないのである．

INKEY$関係を用いて複数の文字を入力する方法を次の例で示してみよう．

```
100 CLS :B$=""
110 PRINT "A=";
120 A$=INKEY$
130 IF A$="" THEN 120
140 IF A$=CHR$(&HD) THEN 180
150 PRINT A$;
160 B$=B$+A$
170 GOTO 120
180 PRINT:PRINT B$
190 END
```

```
A=12345
12345
OK
```

ここで行番号140の&HD(=13)はキャリッジ・リターン（C_R：改行）のアスキー・コードである．

INKEY$関数を用いるとカーソルの点滅がなくなってしまう．このために画面に入力要求を出しても気づきにくい．このため，入力の要求文はブリンク(点滅）させるなどして利用者の目にとまりやすいようにすべきである（45節参照)．

LINE INPUTおよびINKEY$関数の他にINPUT$関数を用いた方法がある．これはINKEY$関数と同様に用いられるが，複数の文字を指定した分だけ同時に入力できるという長所がある．しかし，その指定した分の文字数が入力されるまで値を返さないという特性があるため，利用者には使いづらいものとなる．入力要求の？マークをはずすという本節の主旨だけを考えると，本節ではあまり勧められない用法である．

パソコンメーカのお仕着せに頼らず，作成者の工夫した入力要求を出そう．

49 大文字↔小文字変換

英文字を含んだ文字列の入力に際して，入力のときはその英文字は大文字でも小文字でも構わないが，出力の際は大文字（または小文字）に整理したいと思うことがよくある．（BASIC 命令の入力がその例である．大文字・小文字を入り乱れて入力しても全て大文字に変換される．）ここではその技法を示そう．

覚えておかなければならないことは，アスキーコードで小文字のコードは大文字のコードに$(20)_{16}$（10 進数で 32）を加えれば得られるということである．

〈例〉 A のコード＝$(41)_{16}$（10 進数で 65）

a のコード＝$(61)_{16}$（10 進数で 97）

したがって 1 文字の英小文字が入っている変数 A$を大文字に直して変数 B$にしまうには次の 1 ステップですむ．

B$=CHR$(ASC(A$)−&H20)

次に入力した文字列に含まれる英小文字をすべて大文字に変換するプログラムを考えよう．

```
100 INPUT "A$=";A$
110 IF A$="" THEN 100
120 B$=""
125 FOR K=1 TO LEN(A$)
130   C$=MID$(A$,K,1):CC=ASC(C$)
140   D=0
150   IF CC>=&H61 AND CC<=&H7A THEN D=&H20
160   B$=B$+CHR$(CC-D)
170 NEXT K
180 PRINT B$
190 END
```

```
A$=ArcCos (p/3)
ARCCOS (P/3)
OK
```

行番号150ではaおよびzのアスキーコードが次の値であることを利用している．

$$a=(61)_{16},\quad z=(7A)_{16}$$

以上のことを応用すれば1文の先頭を大文字にする，といった英文ワープロ機能などを容易に実現することができる．また日本語処理で片仮名を平仮名に変換する，といったこともこの論理とまったく同一にできる．

文字処理に親しむと，パソコンの新しい応用の世界が開けてくる．

Memo　文字処理

ワープロ（ワードプロセッサ）の中味はパソコンと同じである．パソコンに文字処理をさせるのがワープロである．文字処理の基本は本節で示したように文字コードを上手に扱うことである．文字コードを上手に扱う手段としてBASICは次のような関数を提供してくれている．

- CHR$ ：指定したコード値をもつ文字を与える
- ASC ：文字コード値を与える
- MID$ ：文字列の中から任意の長さの文字列を与える
- INSTR：文字列の中から指定した文字列を捜し，その位置を与える
- STR$ ：数値を表わす文字列を与える
- VAL ：文字列の表わす数値を与える
- LEN ：文字列の長さを与える

50 特殊な文字の入出力

キーボードからカンマ (,) やクォーテーションマーク (″) を入力しようとするとき，単純に

INPUT A$

とプログラム中にコーディングしたのではだめである．カンマ (,) を入れると

? Redo from start

などというメッセージが出力されるし，クォーテーションマーク (″) を入れると A$にはヌルストリング (″″) が入ってしまう．したがってこのような特殊文字の入力は INPUT 命令では不可能である．BASIC は特殊文字の入力ができるよう次の三つを用意している．

LINE INPUT

INKEY$

INPUT$

ただし改行 (キャリッジリターン C_R) 文字については LINE INPUT では入力できない，次のプログラムは LINE INPUT の用例とその入出力例である．

```
100 INPUT "A$=";A$
110 PRINT "A$=";A$
120 LINE INPUT "B$=";B$
130 PRINT "B$=";B$
140 END
```

```
A$=?″
A$=
B$=″
B$=″
```

次に INKEY$を用いて改行 (キャリッジリターン：$C_R$) 文字を入力することを考えよう．これは単純に次のようにすればよい

```
:
140 A$=INKEY$
150 IF=″ ″ THEN 140
160
:
```

行番号140〜150のループを実行中に改行キーが入力されるとA$にはコード(0D)$_{16}$(=13)が入る．当然(0D)$_{16}$というコードはキャリッジ・リターンのコードである．

このC_Rコードと同様にSTOPキー以外のキーに対する文字コードはすべてINKEY$で読むことができる．

INPUT$関数はINKEY$関数を複数の文字の入力のために拡張したものである．ただしINPUT$では指定した文字数の入力(この文字はSTOPコード以外何でもよい)が終了するまでCPUはWAITの状態になる．

特殊文字の出力はCHR$関数を用いねばならない，たとえばクォーテーションマーク(")を出力したいときに

```
PRINT """
```

ではだめなのである．これはシンタックスエラーとなる．次のように記述する．

```
PRINT  CHR$(&H22)
```

クォーテーションマーク(")のコードは(22)$_{16}$であることを用いている．また画面消去命令の次の用法は有名である．

```
PRINT  CHR$(12)
```

12(16進で0C)というコードは画面消去コード(プリンタでは改ページ)なのである．

特殊文字(制御コード)の入出力ができると，繊細なパソコン制御ができるようになる．

Memo　制御コード(制御文字)

A，B，Cなどの普通の文字に対してコード&H0Dで表現されるC_R(キャリッジ・リターン・コード)などはコンピュータ制御に関係する文字である．このような特殊な文字を制御文字といい，そのコードを制御コードという．

51 多用する文字はファンクションキーに

パソコンの電源投入（リセット）直後，画面にはメーカの選択した文字列がファンクションキーの値として画面に表示されているのが普通である．このファンクションキーの値が簡単に変更できることを知っているとプログラムのコード入力やデータ入力が非常に楽になる．

たとえば通常我々のプログラムはINPUT命令をよく用いるが，多くのパソコンはこれをリセット直後のファンクションキーの値としてもっていない．そこでファンクションキーのN番目にこのINPUT命令を入れておくと楽になる．それには次のコマンドを入力すればよい．

```
KEY  N, " INPUT"
```

以後N番目のファンクションキーを押せばINPUTという文字列がタイプインされるのである．

上位機種のファンクションキーには，LOAD，SAVEがリセット直後にその値としてとられるようになっているが，中以下の機種にはそうなっていないものが多い．ファイル操作をするときどうしてもLOAD, SAVEをよく用いたいのでこれをファンクションキーに定義してみよう．たとえばLOADを考えてみる．この命令は一般に次の形で用いられる．

```
LOAD  " ファイル名"
```

したがって，LOAD"というふうにファンクションキーに入れると便利なことがわかる．上記のようにKEY命令を用いると次のようになる．

```
KEY  1, " LOAD␣""
```

しかしこれではエラーとなる．クォーテーションマーク（"）はKEY命令中に用いることはできない．このとき次のようにする．

```
KEY  1, " LOAD␣" + CHR$(&H 22)
```

ここで $(22)_{16}$ はクォーテーションマークのアスキーコードである．

SAVEについてもまったく同様である．またFILES命令のように，その直後

にリターンキーしか伴わないときには次のように定義すると便利である．

```
KEY 3,"FILES"+CHR$(&H0D)
```

ここで $(\&H\ 0D)_{16}$ は RETURN キーのアスキーコードである．こうすることで３番目のファンクションキーを押すだけでファイルの中味を見る操作が完了するのである．

ちなみに画面からファンクションキーの表示を消すには次の命令を用いる．

```
CONSOLE ,,0
```

このときファンクションキーの中味を見るには次の命令を実行させる．

```
KEY LIST
```

また再び画面にファンクションキーを表示させるには次の命令を実行させる．

```
CONSOLE ,,1
```

ファンクションキーは上手に用いると本当に便利なものである．その内容の変更も，上記のように非常に簡単である．ぜひ多用することをお勧めする．

ファンクションキーを有効に利用すると，入力の手間が大きく省けることに注意しよう．

Memo　ファンクションキー割り込み

上位機種のパソコンはファンクションキーを割り込みキーとして定義できるようになっている．これは次のように用いる．

```
ON KEY GOSUB 2700, 3000, 3200
```

このとき，もしファンクションキー１をこのプログラム実行中に押すとプログラムは中断し行番号 2700 に分岐する．この機能を用いると割り込み機能が大幅に拡充され使いやすくなる．

52 捜しものにはSEARCH, INSTR関数を

プログラム論理として，文字列や配列の中から指定した文字や数を捜すということをしばしば用いる．たとえば，文字列の入った文字型変数 A$の中に＋という文字があるかどうかを調べたりするのは，数式処理では頻繁に用いられる．これを実行するのにいちいち次のようにしたのでは大変である．

```
   :
200  FOR  K=1  TO  LEN(A$)
210     IF  MID$(A$, K, 1)="+"  THEN  230
220  NEXT  K
230  '
   :
```

このようなコーディングでは BASIC による文字処理は不可能である．あまりに時間をとりすぎてしまう．BASIC はそのために次の命令を用意してくれている．

```
   :
200  K=INSTR(A$,"+")
   :
```

また配列の中である整数値をもつ配列要素を捜す論理も上記のように多用される．たとえば生徒の点数が入っている配列 A において満点（100 点）を取った生徒を捜すことを考えよう．(生徒は 300 人いて，配列要素の添字の番号が生徒の受験番号とする．)

```
   :
270  FOR  K=1  TO  300
280     IF  A(K)=100  THEN  PRINT  K
290  NEXT  K
   :
```

しかし SEARCH 命令（標準語ではない）を用いると次のように書ける．

```
    :
270 K=1:I=1
280 WHILE I>0 AND K<=300
290     I=SEARCH(A, 100, K)
300     IF I>0 THEN PRINT I:K=I+1
310 WEND
    :
```

ステップ数は増すが処理速度は速くなる．というのはサーチするループを後者は BASIC の翻訳した機械語で実行するからである．

BASIC は実行速度が遅い言語である．そのため，ある値（文字）を捜すようなループ処理はできることなら，BASIC そのものに任せた方がよい．表題の二つの命令を活用すべきである．

検索には INSTR，SEARCH 関数が意外に役立つものである．

Memo　BASIC 命令の豊富な理由

コンパイル言語（FORTRAN，COBOL 等）では本節の命令のように親切な命令はない．配列から数値を捜したり，文字列から文字を捜したりするのは自分のコーディングで行う．では，なぜ BASIC は豊富な命令を我々に提供してくれているのだろうか？　それは何も単に親切心からだけではない．たびたび述べてきたように BASIC はインタプリタ言語であり，翻訳に手間取る言語なのである．したがって捜しものをするのに FOR～NEXT などのループ処理をしたりするのでは，待ち時間が膨大になる．翻訳時間を節約し，BASIC がコンピュータ言語として十分に役立つようにするには命令を多彩にするしかないのである．我々は BASIC のこの特性をしっかり頭に入れておくべきである．

53 処理速度の調査法

プログラムの処理速度を測定しようとしたり，また比較したりしようとするときいちいちストップ・ウォッチを用意する必要はない．パソコンにはタイマが内蔵されているのである．いま，たとえば次の二つのコーディングのどちらが速いかを比較したいとする．

2＊AかA+Aか？

このとき，パソコンの内蔵タイマを用いて次のようにこの比較を行うことが可能である．

```
100 N=5000
110 TIME$="00:00:00"
120 FOR K=1 TO N
130   X=2*A
140 NEXT K
150 PRINT TIME$
160 TIME$="00:00:00"
170 FOR K=1 TO N
180   X=A+A
190 NEXT K
200 PRINT TIME$
210 END
```

結果は次のように出力される．

```
00 : 00 : 08
00 : 00 : 07
```

この例で分かるように現在のパソコン内蔵のタイマの時刻を知るには

PRINT TIME$

でよく，またこの時刻を合わせたいときには

TIME$=″hh : mm : ss″

とすればよい．ここでhh，mm，ssは時間，分，秒の値である．

上記のプログラミングではパソコンのタイマの時刻をリセットしてしまう．時刻を保存するには，次のようにコーディングするとよい．（上例の行番号 100 から 150 までを書き換える．）

```
100 N=5000
110 T1$=TIME$
120 FOR K=1 TO N
130   X=2*A
140 NEXT K
150 T2$=TIME$
160 '
170 H1=VAL(MID$(T1$,1,2))
180 M1=VAL(MID$(T1$,4,2))
190 S1=VAL(MID$(T1$,7,2))
200 H2=VAL(MID$(T2$,1,2))
210 M2=VAL(MID$(T2$,4,2))
220 S2=VAL(MID$(T2$,7,2))
230 DH=H2-H1:DM=M2-M1:DS=S2-S1
240 PRINT DH;":";DM;":";DS
```

（注：DS 等が負になる場合は読者に考えて頂きたい．）

パソコンにも時計が内蔵されていることを忘れないように．

Memo **タイマ割り込み**

パソコンの上位機種では，表題の制御をサポートしている．これは，次のような形式でユーザが用いられる．

```
ON TIME$="12 : 00 : 00" GOSUB * LUNCH
```

すなわちタイマが正午を告げたとき BASIC は制御を指定されたサブルーチンに移すのである．

54 入力時間を制限するには

CAI(教育用ソフト)やゲーム用のプログラムを作成しようとするとき，入力時間を制限したいときが多い．たとえば2+3の答えを要求するとき，解答者が10秒以上もキーを押さなければ警告メッセージを表示し，入力を禁止して次の処理に移りたくなる．

時間処理としてBASICはタイマ割り込みという手段を我々に提供しているが，BASICの割り込みのサービスよりも入出力割り込みというチャネルの割り込みの方が優先されるため，上記のような場合にはBASICのタイマ割り込みは使用できない．そこで，次のような命令が用意されているパソコンがある．（NEC　PC-8801/9801から採用した．）

INPUT　WAIT　(待ち時間)，プロンプト文；変数名

この命令はINPUTとほぼ同一であるが，待ち時間で指定された長さだけ過ぎたら次の行番号に実行を移すという特性をもつ．例として，小学校用のCAIに用いられる1桁の整数の加法を学習させるプログラムを考えてみよう．

```
100 DEFINT A-Z
110 PRINT:PRINT
120 A=RND*10:B=RND*10
130 PRINT "(ﾓﾝﾀﾞｲ) ";
140 PRINT A;"+";B;"=";
150 INPUT WAIT 100,ANS:GOTO 170
160 PRINT:PRINT "ｼﾞｶﾝﾃﾞｽ":GOTO 200
170 '
180 IF ANS=A+B THEN PRINT "ﾖｸﾃﾞｷﾏｼﾀ":GOTO 110
190 PRINT "ﾏﾁｶﾞｲﾃﾞｽ"
200 BEEP:PRINT "(ｶｲ) ";A+B:GOTO 110
```

このプログラム例を見れば分かるように入力の時間制限にINPUT　WAIT命令が非常に有効である．またこのような命令を用いて入力要求を出すとプログラムが生き生きしてくるのである．

パソコンにも待たされてイライラする権利を与えよう．

55 16進数を画面に出力するには

いま，次の命令を実行してみよう．

```
PRINT "A ノコード="; &H41
```

すると画面には次のように出力される．

```
A ノコード=65
```

すなわち，BASICは16進数でも8進数でもすべて10進数に変換して画面やプリンタに出力するのである．たとえばアスキーコードの値を示したいときは，10進よりも16進で表示したい．ではどうやって16進形式で出力するかを一般的に示そう．

変数Iに入っている整数を16進数形式で表示するには，次のようなコーディングをすればよい．

```
:
140 I$=HEX$(I)
150 PRINT I$
:
```

すなわち，Iという値をまず16進の文字列に変換（HEX$(I)）し，それを出力するのである．画面からの入力は16進数形式が許されている．INPUT命令に対してたとえば，&H41と入力してもきちんと正しく入力される．

8進数による画面への表示もまったく同一である．ただし，こんどは変換関数としてOCT$を用いることになる．

```
:
140 I$=OCT$(I)
150 PRINT I$
:
```

なお最後にここで述べたことは，そのままプリンタへの出力にも応用できる．

16進数の方がわかりやすい数値は16進数で出力しよう．

56 配列の添字を1から始めてメモリ節約

BASICの配列の添字は，0から始められるのはBASIC言語を非常に使いやすくしている．しかし，反面添字の値として0をとらないようなプログラムにおいてはその分だけ未使用となりメモリを浪費してしまう．特に2次元配列や3次元配列においては深刻である．そこで上位機種のパソコンでは配列の添字を1から始めるように変更できる命令を用意している．それが次の命令である．

OPTION　BASE　0または1

0と記すと添字は0から，1と記すと添字は1から始まる．

A (0,0)	A (0,1)	A (0,2)	…	A (0,4)
A (1,0)				⋮
⋮				⋮
⋮				⋮
A (4,0)	…	…	…	A (4,4)

〈例〉　OPTION BASE 0
　　　の配列 A (4, 4)

A (1,1)	A (1,2)	…	A (1,4)
A (2,1)			⋮
⋮			⋮
A (4,0)	…	…	A (4,4)

〈例〉　OPTION BASE 1
　　　の配列 A (4, 4)

この二つの例からも分かるようにずい分と使用メモリが少なくなる．たとえば実数型配列変数においてA (10, 10)をとると次のようなメモリ計算になる．

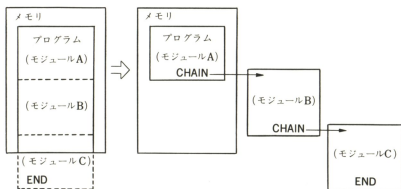
```
OPTION  BASE  0              OPTION  BASE  1
DIM  A (10, 10)              DIM  A (10, 10)
  :                            :
11×11×4=484バイト            10×10×4=400バイト
```

2割以上のメモリ節約になるのである．配列においてスタートを0にするか1にするかは大きな違いとなることを理解してもらいたい．

大きな配列を用いるときには，少しでもメモリの節約を考えよう．

57 大きなプログラムは分割してCHAIN

大きなプログラムを作成するとそれがパソコンのメモリに入りきれなくなることがよくある．メモリに入りきれなくなる理由は二つある．(1) プログラムが長すぎる，(2) データ領域が大きすぎる，この節では(1)の場合を扱う．

メモリに格納し切れないほどの長さをもつプログラムは当然モジュール化がなされていなくてはならない（2節参照）．そしてもしこれがなされているなら，プログラム入力または連結途上でメモリオーバになったとしてもあわてることはない．このプログラムの実行を分割すればよいのである．それがCHAIN命令である．

細かい仕様は，マニュアルを調べてもらいたいが，上の図でその使い方の概略は理解されるであろう．

我々は，大きなプログラムを次のように作成するのが普通である．まずモジュール化した設計をプログラムに与え，その各モジュールを別々に作成し，また別々にテストする．各モジュールのデバックが完了したなら始めてCHAIN命令を用いて結合し実行させるのである．こうすることで複数の人間によるプログラム開発が容易となり開発期間の短縮がなされるのである．

パソコンにおいて大きなプログラム作成時にCHAIN命令は不可欠である．

58 FOR～NEXTの増分は1だけではない

BASICの入門書には，FOR～NEXT命令の説明においてSTEP指定の解説を省いているものが多い．そのためか，しばしば次のようなコーディングを見かける．

```
270 FOR K=1 TO 10
280   I=10+20*K
290   LINE (0,I)-(639,I)
300 NEXT K
```

これは画面に罫線を入れるものだが，やはり次のようにすべきである．

```
270 FOR K=30 TO 210 STEP 20
280   LINE (0,K)-(639,K)
290 NEXT K
```

FOR～NEXT命令の一般形は次のように書ける．

```
FOR K=I TO J STEP D
 ∫
NEXT K
```

ここで銘記しておくべきことは，I, J, Dの値を何にとってもよいということである．FORTRANやCOBOLに慣れた人はついI, J, Dの値が正の整数でなければならないと思い込みがちだが，そうではない．小数値および負の値でもよいのである．

ここでは，上記の形式の変数I，J，Dの値として小数値をとったときの注意点を述べよう．32節で記載したが，もう一度次のプログラムを見てみよう．

```
100 S=0
110 FOR X=0 TO 1 STEP .1
120   S=S+X
130 NEXT X
140 PRINT S
150 END
```

これは，0.1＋0.2＋……＋1.0の値を求めるつもりで作成されたプログラムであるが，正解を出さない．すなわち，正解は5.5となるべきところを4.5と出力し

てしまうのである．これはコンピュータのデータ表現の構造を無視したために起きた誤りなのである．(32 節参照.)．

我々は FOR〜NEXT を次のように記述したとき，どうしても右のように解釈してしまう．

FOR X=I TO J STEP D　　X が I から始まり D ずつ増えてゆき値が J に（もしくはそれより大きく）なったときループが終わる．

したがって，上のプログラムでは当然 1.0 も和 S の中に含まれると解釈してしまうのである（しかし現実には上記のようにそうなってはいない！）．

この解釈を生かすには，我々は次のように FOR〜NEXT 命令をコーディングすべきである．

D の値は整数または$\pm\frac{1}{2^N}$（N=1,2,……）の形にせよ．

このことさえ守っていれば，FOR〜NEXT の命令を素直に解釈しても間違いは生じなくなる．

（例）　FOR K=1 TO 3 STEP 1/8

D の値が与えられていて変更できないときには，次のように工夫する．すなわち上記の J の値として次のように指定するのである．

（例）　FOR K=1 TO J+.1 * D STEP D

こうすることで，簡単に上記のような誤りから逃れられるのである．しかし，このようにコーディングするとプログラムが美しくなくなるという欠点をもってしまうのは否定できない．

STEP 値が小数のときには慎重さが要求される．

59 ループ計算にはWHILE～WENDも便利

ループ計算を BASIC の標準語で実行させようと思うと，FOR～NEXT 命令の利用がある．しかし，上位機種では，ある条件が満たされている間ループ計算を行え，という指定ができる命令が備えられている．NEC PC-8801／9801 はそれを WHILE～WEND 命令で実現している．

たとえば次の例を考えよう．自然数の 4 乗の和

$$S=1^4+2^4+\cdots\cdots+N^4$$

でその和 S が始めて 1000 を越える N の値を求めるプログラムを考えてみよう．GOTO 命令はなるべく用いないという要請（14 節参照）からこれを FOR～NEXT を用いて記述してみよう．

```
100 S=0
110 FOR N=1 TO 10000
120   S=S+N^4
130   IF S>1000 THEN 150
140 NEXT N
150 PRINT "N=";N
160 END
```

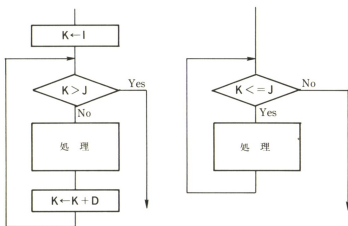
行番号 110 で K の終りの値を 10000 としたのは，これ位の大きな値を指定しておけば十分であろうという予想があるからである．もちろんその予想は正しいがコーディングの美しさに欠けるところがある．このようなとき，WHILE～WEND は非常に便利である．

```
100 S=0:N=0
110 WHILE S<=1000
120   N=N+1
130   S=S+N^4
140 WEND
150 PRINT "N=";N
160 END
```

これなら前のプログラムの FOR～NEXT に指定した 10000 のような曖昧さを残さない．このように，WHILE～WEND は区間の指定ができないような繰り返し計算に非常に便利であることが分かるであろう．

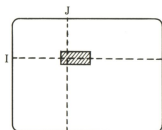
WHILE～WEND は，PL/I のような高級言語のもつ便利な命令を BASIC でもサポートしようという観点から生まれた命令である．BASIC は繰り返し用の計算命令として FOR～NEXT も用意しているが各々の特徴を生かして利用してもらいたい．

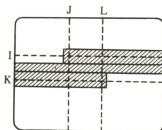
この WHILE～WEND の記述においても，当然 12 節で解説した段づけの技法を生かさねばならない．逆にいって，この段づけをきちっとやることで FOR～NEXT と同様 WHILE～WEND 命令が生きてくるのである．

最後に，FOR～NEXT と WHLILE～WEND の処理の違いをフローチャートに示しておこう．

FOR～NEXT，WHILE～WEND を用いて GOTO 命令を追放しよう．

60 テキスト画面の一部を消すには

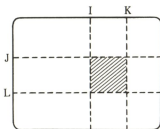
画面のすべてを消すには，CLS という命令がある．しかし，文章などの編集をするプログラムにおいては，消しゴムで文書の一部を消すのと同様に画面の一部を消したいことがある．また CAI 用ソフトの作成において画面に書いた答えをそこだけ再び隠す，といった技法をよく使う．

画面上の数文字を消すなら次の命令を用いればよい．

ここで PRINT 命令で書くスペース（空白）の字数は消したい文字の字数に一致させる．

消したい文字数が多いときや，消したい部分が複数行にわたるときには次のような命令を用いればよい．

ここで W とは画面の 1 行に入る最大文字数である．

日本語の文章のように縦書きで画面を利用しているとき，縦方向に画面の一部を消したいことがある．もちろん上記の方法を組み合わせればこのことは可能だがめんどうである．NEC PC-8801/9801 などには次のような命令が備えられている．

COLOR@　(X1, Y1) − (X2, Y2), FC

これは画面のテキスト座標（X1, Y1）と（X2, Y2）とを結ぶ線を対角線とする水平な長方形の内側の文字の色を変える命令である．通常テキスト画面のバックグランドの色は黒であるからそのカラーコードに文字の色をあてれば結局画面からその部分の文字がすべて消えてしまう．

ここで FC=0 は黒のカラーコードである．

白黒画面で上の機能を実現するには次のようにする．

COLOR@　(I, J) − (K, L) , 1

FC=1 は白黒モードのときその部分にある文字を隠す機能コードである．

Memo　**消すと隠す**

CLS 命令や空白で上書きする方法は画面上の文字を消すことになる．しかし下地の色と同じ色に書かれている文字の色を合わせる方法は隠すことに相当する．すなわち，下地の色を変えると前の文字が浮かび上がってしまうのである．この辺のことに注意が必要である．

61 色々な関数は組み込み関数の組み合わせで

たとえば常用対数など，有名な関数がすべてBASICの組み込み関数として提供されているわけではない．しかし，ほとんどの有名な関数はBASICの組み込み関数でサポートできる．

たとえば常用対数をみてみよう．これは次のようにLOG（X）からみちびかれる．

（常用対数）　LOG（X)/LOG（10)

これは数学で用いられる底の変換公式を用いている．

また立方根 $\sqrt[3]{x}$ は指数演算を用いて次のように記述できる．

（立方根）　X^(1/3)

応用上一番大変なものが逆三角関数である．BASICの数値関数としてATN（X）があるがこれは次の機能をはたす．

tan Y＝X．を満すYで $-\frac{\pi}{2}$ から $\frac{\pi}{2}$ の値を満すもの

これを用いると

(1)　sin Y＝Xを満すYで $-\frac{\pi}{2}$ から $\frac{\pi}{2}$ の値を満すものを求める関数ASIN（X）（逆正弦関数）は

$$\text{ASIN (X)}=\begin{cases}\text{ATN (X/SQR (1-X * X)) (X}\neq\pm1) \\ \frac{\pi}{2}\ (X=1) \\ -\frac{\pi}{2}\ (X=-1)\end{cases}$$

(2)　COS Y＝Xを満すYで0から π の値を満すものを求める関数ACOS（X）（逆余弦関数）は

$$
\text{ACOS (X)} \begin{cases} \text{ATN (SQR (1-X * X)/X)} & (0<X\leqq 1) \\ \text{ATN (SQR (1-X * X)/X)}+\pi & (-1<X<0) \\ \frac{\pi}{2} & (X=0) \\ \pi & (X=-1) \end{cases}
$$

BASICの組み込み関数を用いて色々な重要な関数を出すには，やはり数学の知識が必要であろう．

使いたい関数がBASICに用意されていなくてもあわてないように．

Memo　**BASICの数値関数**

BASICが提供する数学上の有名な関数は本節で述べたLOG，指数演算，ATN以外に次のようなものがある．

SIN (X)	：角X（ラジアン）の正弦値を与える
COS(X)	：角X（ラジアン）の余弦値を与える
TAN(X)	：角X（ラジアン）の正接値を与える
ABS(X)	：Xの絶対値を与える
INT (X)	：GAUSS関数［X］である
FIX (X)	：小数部を切り捨てる
SGN(X)	：符号を与える
SQR(X)	：平方根を与える
EXP(X)	：指数関数（底がe=2.718281…）である

これらを上手に組み合わせることで色々と有益な関数が生れるのである．

62 GOSUB命令の再帰的用法

次のプログラムを見てみよう．

```
100 N=3:DIM A(N)
110 D=2:A(1)=1:I=N
120 GOSUB *SUB
125 FOR K=1 TO N:PRINT A(K):NEXT K
130 END
140 *SUB
150 IF I=1 THEN 200
160 I=I-1
170 GOSUB *SUB
180 A(I+1)=A(I)+D
190 I=I+1
200 RETURN
```

これは，初項1，公差2の等差数列（すなわち奇数列）を始めから3個列挙するプログラムなのである．

```
RUN
1
3
5
OK
```

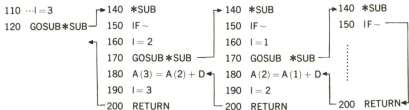
これは次のプログラムと等価である．

```
100 N=3:DIM A(N)
110 A(1)=1:D=2
120 FOR K=1 TO 2
130   A(K+1)=A(K)+D
140 NEXT K
150 FOR K=1 TO N:PRINT A(K):NEXT K
160 END
```

このように簡単に書けるプログラムをなぜはじめのように書くのかというと，このようなプログラムの記述方法に慣れておくと，複雑な論理を処理するときに，しばしば役に立つからである．特に何段階もの同一論理の入れ子処理をす

るときに，その有用性を発揮する．

はじめのプログラムを説明しよう．このような GOSUB 命令の用い方を**再帰的**（または**回帰的**）な用法といい，この命令を用いたプログラムを再帰的（または回帰的）なプログラムという．

＊SUB というラベルで始まるサブルーチンは，まずメインルーチンの行番号 120 から呼び出される．そして，このサブルーチンに実行が移されると，その中に再び自らを呼ぶ命令（行番号 170）が入っているのである．これを文章で説明することは冗長となるので，次の図で理解してもらいたい．

BASIC はこのような再帰的プログラミングを得意としない．しかしこの論理を知っていることは一つの強力な武器を手にしたことと同じになる．ぜひとも覚えておいてもらいたい．なお，このような論法を得意とするコンピュータ言語がいくつかある．その中で有名なものが，LISP や PASCAL である．

これからコンピュータを学ぶ者は，再帰的な考え方に慣れておくことが必要である．

Memo **回帰的なプログラムと人工知能**

近年コンピュータの人工知能化がいわれているが，これを実現する有力な表現となるものの一つに本節で扱った回帰的（再帰的）な表現がある．これからコンピュータをしっかり学んでゆこうと思う方にはこの論理が必須であろう．

63 RND関数で正規乱数をつくる

統計学の専門家でもない限り，乱数を仕事で利用することはなかったであろう．しかし，コンピュータを使い出すと，意外に多くの場面で乱数を活用するようになる．たとえば，データのモデルを大量に作ってみたり，ゲーム等で，デタラメに図形を動かしてみたり，10円玉や，サイコロを振ったりするシュミレーションに使ったり実に様々である．

コンピュータを用いて乱数を発生させるにはRND関数を用いればよく，RND(I)なる命令を実行すると0以上，1未満の乱数が一つ作られることになる(ただし，Iには具体的な数値が入り，その符号などによって発生される乱数が変わってくるのでマニュアルをよく読んでほしい)．したがって，0以上，N以下の整数をデタラメに作成するには，INT((N+1) * RND(1)) と命令し，−N以上，N以下の整数をデタラメに作成するには，INT((2N+1) * RND(1))−N と命令すればよい．また，サイコロをふるシュミレーションに使うのであれば，RND(1)の値が(i−1)／6以上，i／6未満のときにはサイコロのiの目が出たと解釈すればよいのである．これらの使い方は，RND関数が0以上，1未満の数値をデタラメではあるが，かたよりなく一様に発生させることを前提に成立しているのである．もちろん，そうでないと一様乱数の意味がなくなってしまう．

この節では，この一様乱数を発生させるRND関数を用いて，正規乱数を発生させるプログラムを紹介することにしよう．そのためのプログラムとして次のようなものが考えられる．

```
100 N=1000:DIM A(N)
110 K=20:K1=SQR(12/K):K2=K/2
120 ME=50:SD=10
130 FOR I=1 TO N
140   FOR J=1 TO K
150     Z=Z+RND(1)
160   NEXT J
170   X=SD*(Z-K2)*K1+ME
180   A(I)=X
190   Z=0
```

```
200 NEXT I
```

このプログラムは平均がMEで，標準偏差がSDとなるような正規分布をなすデータをN個作成するものである．ここではME=50, SD=10, N=1000として処理している．実際につくられた1000個のデータを分布グラフで示すと次のようになっている．

```
  0 -(   0)I
  5 -(   0)I
 10 -(   1)I
 15 -(   0)I
 20 -(   8)I
 25 -(  22)I**
 30 -(  54)I*****
 35 -(  85)I********
 40 -(154)I***************
 45 -(180)I******************
 50 -(200)I********************
 55 -(146)I**************
 60 -(  78)I*******
 65 -(  53)I*****
 70 -(  13)I*
 75 -(   5)I
 80 -(   1)I
 85 -(   0)I
 90 -(   0)I
 95 -(   0)I
100 -(   0)I




ﾍｲｷﾝ= 49.6477                ﾋｮｳｼﾞｭﾝﾍﾝｻ= 10.297
```

この分布グラフや，実際に作られたデータの平均，標準偏差を得るにあたっては，先のプログラムに，次のプログラムを追加して処理をしてみた．ただし，＊印は10個分の度数を表現しているものとする．

```
800 '
810 D=5:H=INT((ME+SD*5)/D):DIM B(H)
820 FOR I=1 TO N
830   T=INT(A(I)/D):B(T)=B(T)+1
840 NEXT I
850 FOR I=0 TO H
860   PRINT USING"###--(###)I";I*D;B(I);
870   IF B(I)=0 THEN PRINT :GOTO 900
880   FOR J=1 TO B(I)/10:PRINT "*";:NEXT J
890   PRINT
900 NEXT I
910 '
920 FOR I=1 TO N
930   S1=S1+A(I)
940   S2=S2+A(I)*A(I)
950 NEXT I
960 M1=S1/N
970 S=SQR(S2/N-M1*M1)
980 PRINT "ﾍｲｷﾝ=";M1,"ﾋｮｳｼﾞｭﾝﾍﾝｻ=";S
```

正規乱数を作るプログラムは，基本的には，一様乱数をいくつか加えて作った新たな数は正規分布に近くなるという性質を用いてプログラミングされているのであるが，その加え合わせる回数をK回としたのである．Kは6以上の値をとれば十分であることが知られている．このプログラムではK=20としてある．

Memo **RND関数**

この関数を上手に用いると，パソコンの応用の世界は大きく広がる．ゲーム，CAI，テスト・データ作成などに非常に便利なものである．BASICに組み込まれた一つの関数としてだけとらえないで頂きたい．

64 BEEP命令で音色を出す

BEEP命令で鳴らされるブザーの音はピーと単色の音が鳴り続けるだけであじけないが，少し工夫すると何種類かの音が作れる．たとえば次のプログラムを実行してみよう．

```
100 N=1000
110 FOR K=1 TO N
120   BEEP(1)
130 NEXT K
140 FOR K=1 TO N
150   BEEP(1):BEEP(0)
160 NEXT K
170 FOR K=1 TO N
180   BEEP(1):BEEP(1):BEEP(0)
190 NEXT K
200 FOR K=1 TO N
210   BEEP(1):BEEP(0):BEEP(0)
220 NEXT K
230 END
```

音を聴いてみれば明らかに音色が違っている．すなわち，ブザー音のスイッチのオン（BEEP(1)），オフ（BEEP(0)）の組み合わせでいろんな音を出せるのである．

凝った音を出すには機械語で書くしかないが，BASIC命令でも上のような工夫を加えることで，5〜6種の音を美しく出せるのである．音の効果は画面表示と同様あるいはそれ以上にコンピュータを使いやすくしてくれるものである．ブザー音は一通りである，といった誤解は捨てるべきである．

BEEP命令はパソコンの発する声となる．

第5章

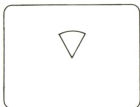
グラフィック命令活用法

この章ではグラフィック命令の便利で有名な使い方を示そう．BASICはグラフィックについての強力な命令を多く有している．したがって，それらを知らないとBASICを利用する価値がなくなる．以下に述べることを参考にしてBASIC命令のグラフィックへの応用を次第に開拓していってもらいたい．

65 扇形はCIRCLE命令で

しばしば見うけられる扇形を描くコーディングは次のようなものである.

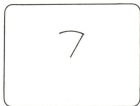
```
100 P=3.14159:T1=P/3:T2=2*P/3
110 CX=320:CY=100:R=100:W=.5
120 CIRCLE (CX,CY),R,,T1,T2
130 LINE (CX,CY)-(CX+R*COS(T1),CY-R*W*SIN(T1))
140 LINE (CX,CY)-(CX+R*COS(T2),CY-R*W*SIN(T2))
```

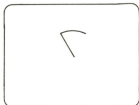
これを実行すると次のような扇形が画面に描かれる.

しかし, もしこれと同一の出力結果を出すためだけなら, 次のような簡単な命令を BASIC は用意してくれている. すなわち行番号 120〜140 が 1 ステップですむのである.

120　CIRCLE (CX, CY), R,, −T1, −T2

すなわち, 弧を描くための角度指定を, その値のままマイナスをつけるとその弧に半径が付け加えられる. すなわち弧が描かれるのである.

たとえば一方だけを負にしてみよう. (行番号は既掲のプログラムに従う).

(1)　T1 にマイナスをつける

120　CIRCLE (CX, CY), R,, −T1, T2

(2) **T2 にマイナスをつける**

```
120 CIRCLE (CX, CY), R,, T1, -T2
```

CIRCLE 命令のこの便利な用法をしっかり覚えておくことは，コーディングを楽にし，処理速度を向上させてくれる．

CIRCLE 命令は円だけを描くのではない！

応用例として，円グラフを描くプログラムとその出力結果を示しておこう．（これは 20 節で既に記載してある）．

```
100 CLS 2
110 N=5:P=3.14159:TO=5*P/2
120 CX=320:CY=100:R=100
130 FOR K=1 TO N
140   READ W:DT=2*P*W/100
150   T1=TO:IF TO>2*P THEN T1=TO-2*P
160   CIRCLE (CX,CY),R,,-TO,-(TO-DT)
170   TO=TO-DT
180 NEXT K
190 DATA 30,25,20,15,10
200 END
```

66 長方形はLINE命令で

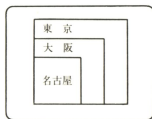
次のプログラムを見てみよう．

```
300 X1=100:Y1=50:X2=100:Y2=100
310 X3=400:Y3=100:X4=400:Y4=50
320 LINE (X1,Y1)-(X2,Y2)
330 LINE (X2,Y2)-(X3,Y3)
340 LINE (X3,Y3)-(X4,Y4)
350 LINE (X4,Y4)-(X1,Y1)
```

これは画面に長方形を描いている．しかし，長方形を描くだけなら行番号 300 ～350 は次の 2 行ですむ．

```
300 X1=100:Y1=50:X3=400:Y3=100
310 LINE (X1,Y1)-(X3,Y3),,B
```

BASIC の命令数を減らすことはプログラムの効率化に大いに寄与する．LINE 命令のこの使い方を上手に利用すると色々なところでグラフィック上での効率化がなされることが可能である．

この LINE 命令を用いた使い方の一つを紹介しよう．下記出力例のように，正方形の面積で，たとえば人口の大きさを示すグラフを作るとする．

これを上記 B 指定を使用しないで作ろうとすると次のようになる．

```
100 CLS 3 :N=3
110 OX=100:OY=150:LX=200:LY=100
120 FOR K=1 TO N:READ R(K):NEXT K
130 LINE (OX,OY)-(OX+LX,OY)
140 LINE (OX,OY-LY)-(OX,OY)
150 FOR K=1 TO N
```

```
160    XG=OX+R(K)*LX:YG=OY-R(K)*LY
170    LINE (XG,OY)-(XG,YG)
180    LINE (XG,YG)-(OX,YG)
190 NEXT K
200 DATA .5,.7,1
210 END
```

ここで出力例に示した都市名はプログラミングしていない．これをB指定を用いて記述すると次のように簡潔になる．

```
100 CLS 3 :N=3
110 OX=100:OY=150:LX=200:LY=100
120 FOR K=1 TO N:READ R(K):NEXT K
130 FOR K=1 TO N
140    XG=OX+R(K)*LX:YG=OY-R(K)*LY
150    LINE (OX,OY)-(XG,YG),,B
160 NEXT K
170 DATA .5,.7,1
180 END
```

グラフィック処理は画面に水平・垂直の線をしばしば描くが，このLINE命令のB指定を上手に用いると，この正方形のグラフの例のようにプログラムが非常に簡単になり，処理効率も向上する．

LINE命令は直線だけを描くのではない．

Memo　**LINE (X1, Y1)－(X2, Y2), C1, BF, C2**

B指定のところをBFとし，C1, C2に適当な自然数値（0〜7）をとると，パソコンは次の命令と同一のことをしてくれる．

```
10 LINE (X1, Y1)-(X2, Y2), C1, B
20 PAINT((X1+X2)/2, (Y1+Y2)/2), C2, C1
```

すなわち，描いた長方形内部をコードC2に対応する色で塗りつぶしてくれるのである．このことも覚えておくとグラフィック処理が非常に簡潔になってゆく．

67 拡張CIRCLE命令で回転体を描く

NEC PC-8801/9801等ではCIRCLE命令に次のような機能を追加している．

CIRCLE (X, Y) , R, C1,,, W, F, C2

ここで，Rは半径，C1は円の色，Wは偏平率である．拡張機能であるF，C2指定は，このCIRCLE命令で描かれた図形の内側を色コードC2で塗りつぶせというものである．

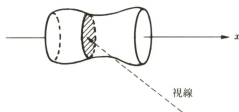
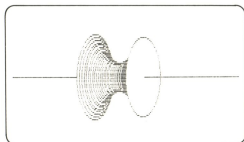
3次元グラフィックについては，他書にゆずることにするが，このCIRCLE命令を用いると，3次元の回転体が簡単に描けることは特筆に値しよう．その例として数学上の関数

$$y = x^2 + 2 \qquad -2 \leqq x \leqq 2$$

をx軸の回りに回転してできる立体図形を画面に描いてみよう．まずプログラムを示そう．

```
100 CLS 3
110 OX=320:OY=100:UX=20:UY=5
120 LINE (0,OY)-(640,OY)
130 A=-2:B=2:DEF FNF(X)=X*X+2
140 DO=.1:GOSUB 160
150 END
160 '
170 FOR X=A TO B STEP DO
180   R=ABS(FNF(X))*UY
190   CIRCLE (OX+UX*X,OY),R,,,,1,F,0
200 NEXT X
210 LINE (OX+UX*X,OY)-(640,OY)
220 RETURN
```

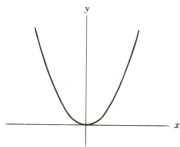
これだけの短いプログラムで立体図がす速く描けるのはおもしろい．また回転体のグラフは応用範囲が広いのでこの技法は知っていると便利である．

この技法の論理は下図で理解されるであろう．

すなわち，斜め無限遠方から見ればきり口の円はだ円となる．それを何枚も重ねてゆけば輪郭は回転体となる．ただし，重ねるときに次のようにすることで3次元の現実感が増す．

すなわち，手前に描いただ円に重なる以前のだ円は，拡張CIRCLE命令のF指定で消してゆくのである．

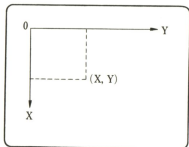
3次元グラフィックも工夫すると意外に簡単になることがある．

68 座標変換とWINDOW－VIEW命令

数学におけるグラフをディスプレイ画面に描こうとするとき，まず我々はそのグラフを紙または頭の中に描くはずである．たとえば，関数 $y=x^2$ のグラフなら次のような図を頭に思い浮かべる．

これをディスプレイ画面にどのように描くかが第2段階である．それには次の操作を伴う．

(1) 原点を画面のどこに置くかを決める．

(2) 数学上の単位1を画面の何ドットにするかを決める．

いま640×200ドットの画面に $y=x^2$ のグラフを描いてみよう．この画面には次のような物理座標がついている．

ここで，X，Yは各々0から639，0から199までの整数値をとる．上記(1)，

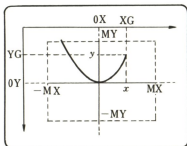
(2)の操作をここで次のようにしてみよう．

(1) 数学上の原点を画面の中央，すなわち物理座標で（320，100）の位置にする．

(2) 数学上の単位1に横方向（X方向）は16ドット，縦方向は8ドットを対応させる．

この選択によって次のプログラムが作成される．

```
100 CLS 3
110 OX=320:OY=100:UX=16:UY=8:MX=10:MY=10
120 LINE (OX,OY-UY*MY)-(OX,OY+UY*MY)
130 LINE (OX-UX*MX,OY)-(OX+UX*MX,OY)
140 DEF FNF(X)=X*X
150 FOR X=-MX TO MX STEP .1
160   Y=FNF(X)
170   IF Y<-MY OR Y>MY THEN 200
180   XG=OX+UX*X:YG=OY-UY*Y
190   PSET (XG,YG)
200 NEXT X
210 END
```

これは次のようなイメージでプログラミングされている．

すなわち，行番号180において

XG=OX+UX＊X

YG=OY−UY＊Y

は画面の物理座標（XG, YG）と数学上の座標（X, Y）とを結びつける**座標変換**の式なのである．

慣れた者にとっては，この論理は分かりやすく，また応用上有益である．しかし始めてグラフィックを扱う人々には難解に見えてしまうだろう．最近の上位機種のパソコンは，グラフィック機能が充実し，このような座標変換をしなくてもすむような命令がつけられている．これを NEC PC-8801/9801 で説明してみよう．

このパソコンのグラフィック機能を用いると，座標変換の式はいらなくなる．頭（または紙上）に描くようにプログラミングすることを可能にしている．すなわち，数学上の座標平面のどこからどこまでを画面のどこに出力するのか，という指定を最初にすることでこのことが実現できるのである．

数学上の座標平面のどこからどこまでを画面に写し出すかの指定が

WINDOW (X1, Y1)－ (X2, Y2)

である．これによって，数学上の座標平面上の点 (X1, Y1)，(X2, Y2) を対角とした長方形の内部が画面上に映し出されることになる．

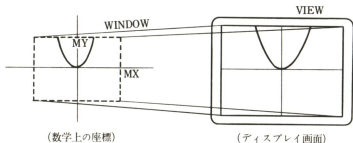
画面のどこに映し出すかを指定するのが

VIEW (XG1, YG1)－ (XG2, YG2)

である．これによって画面上の物理座標 (XG1, YG1)，(XG2, YG2) を対角とする長方形の内部にグラフが描かれることになる．

さっそくこの二つの命令を用いて，前記と同一の内容をもつプログラミングをしてみよう．

```
100 CLS 3
110 KO=2*200/640
120 MX=10:MY=KO*MX
130 WINDOW (-MX,-MY)-(MX,MY)
140 VIEW (0,0)-(639,199)
150 LINE (-MX,0)-(MX,0)
160 LINE (0,-MY)-(0,MY)
170 DEF FNF(X)=X*X
180 FOR X=-MX TO MX STEP .1
190   Y=FNF(X)
200   IF Y<-MY OR Y>MY THEN 220
210   PSET (X,-Y)
220 NEXT X
230 END
```

最初のプログラムに比べて，グラフを実際に描く FOR～NEXT の内側の処理が非常に我々の用いる数学に近づいたことがよく分かると思う．また複雑なグラフを描かせるとき，座標変換をしない分だけ計算スピードが大きくなり，我々の待ち時間が短縮されるのである．このプログラムの内部処理の原理を下図に示してみよう．

グラフィック処理は，BASIC 言語で記述すると時間がかかる．したがって，なるべく BASIC が提供してくれているこの WINDOW-VIEW 命令などを用いて，処理速度を向上すべきである．また，これらの命令を用いることで，ほとんどハードウェアを意識することなく数学の論理だけでプログラムが記述できる，というメリットも必然的に生まれてくるのである．

グラフィックは BASIC の豊富なグラフィック機能に頼ろう．

69 好みの色を出すには

ブラウン管にすぐに表示できる色は8色である．しかし，その8色を上手に混ぜ合わせることで，8色よりもはるかに多くの色を出すことができる(8色以外の色のことを**中間色**と呼んでいる.).

たとえば，円の内側をピンクで塗りつぶしたいとする．ピンクは基本の8色の中には存在しない．そこで次のような命令を用いるのである．

```
100 SCREEN 0,0
110 CLS 3
120 CIRCLE (320,100),100,7
130 TILE$=CHR$(&HAA)+CHR$(&HFF)+CHR$(&HAA)
140 PAINT (320,100),TILE$,7
```

実際RUNさせてみると,確かに円の内側がピンクで塗られる.行番号130の文字変数TILE$を定義する16進数の値を変えてゆくと，どんどん新しい色が出現してくるであろう．

このグラフィックの用い方は，目の錯覚を利用している．すなわち，上例の場合だと画面に640×200ドットの点があるわけだが,その各ドットの色を変えると，全体として基本の8色とは違った色が出現するのである．

NEC PC-8801/9801では，色を混合する基本単位を画面上の横8ドットにとっている．この8ドットの各々に基本色の8色を任意に対応させることによって，色々な配色を実現させるのである．上記プログラム例では，変数TILE$でその8ドットの各々のドット色を定義している.最初の16進数は青,次の16進数は赤，そして最後の16進数は緑に対応し，各16進数を2進数にしたときの8ビットの各桁を青赤緑の点滅のON-OFF（1がON, 0がOFF）としているのである．上の例でなぜピンクが出せたかは，次ページの図で分かるであろう．

この機能を拡張すると色々な色と模様で画面を塗りつぶせるようになるのだが，ここではとにかく8色の基本色以外にもはるかに多くの配色が出せることを示しておこう．

	16進数	2進数	ドットパターン
青 （第1プレーン）	AA	10101010	○●○●○●○●
赤 （第2プレーン）	FF	11111111	●●●●●●●●
緑 （第3プレーン）	AA	10101010	○●○●○●○●
			赤白赤白赤白赤白 ピンク

画面上に無限の色の変化をつけられることを覚えておこう

Memo 色の三原色

光の色の三原色は赤，青，緑である．これらを組み合わせることで無限の色の種類を作ることができる．

多くのパソコンでは簡単に指定できる色として8色を用意している．

（例） PAINT (X, Y), 3, 7

これは，赤，青，緑をディスプレイ画面上の1ドットに割り振る方法の数に等しい。

青のドット	赤のドット	緑のドット	出てくる色
○	○	○	黒
●	○	○	青
●	●	○	紫
●	○	●	水色
○	●	○	赤
○	●	●	黄色
○	○	●	緑
●	●	●	白

○はOFF　●はON

この表から8色（$=2^3$）の色が容易に出せることがわかる．

70 図に模様をつける

画面の図に種々の色を塗ることは，BASIC の PAINT 命令で実行できる．たとえば，円を赤く塗りつぶすには次の命令ですむ．

CIRCLE (320, 100), 100, 7

PAINT (320, 100), 2, 7

このように図形を色で塗りつぶすのと同じように，図形を模様で塗りつぶしたくなることも多い．たとえば縞模様にしたり，チェック模様にしたりするのである．このことは PSET 命令を用いて我々が自分でプログラミングできるが大変である．上位機種では，そのため次のような命令でこの機能を提供してくれている．ここで，NEC PC-8801/9801 を例にとって解説しよう．

PAINT (X, Y), TILE$, C

ここで C は境界の色である．また文字変数 TILE$（この名前は何でもよい）には模様のパターンを入れておく．たとえば，次のような縞模様で円を埋めたいときには，その下のようなコーディングをすればよい．(変数 N が縞の間隔を規定している.)

```
100 SCREEN 0,0:CLS 3
110 CIRCLE (320,100),100,7
120 TL1$=CHR$(&HFF)+CHR$(&HFF)+CHR$(&HFF)
130 TL2$=CHR$(&HO)+CHR$(&HO)+CHR$(&HO)
140 N=3
150 TILE$=""
160 FOR K=1 TO N:TILE$=TILE$+TL1$:NEXT K
170 FOR K=1 TO N:TILE$=TILE$+TL2$:NEXT K
180 PAINT (320,100),TILE$,7
190 END
```

また，下図のようなチェック模様を出すには，その下のようにコーディング

する（変数Nはチェックの細かさを規定する.）.

```
100 SCREEN 0,0:CLS 3
110 CIRCLE (320,100),100,7
120 TL1$=CHR$(&HF)+CHR$(&HF)+CHR$(&HF)
130 TL2$=CHR$(&HF0)+CHR$(&HF0)+CHR$(&HF0)
140 N=3
150 TILE$=""
160 FOR K=1 TO N:TILE$=TILE$+TL1$:NEXT K
170 FOR K=1 TO N:TILE$=TILE$+TL2$:NEXT K
180 PAINT (320,100),TILE$,7
190 END
```

どうしてこのような方法で模様が描けるかは，各自パソコンのマニュアルを調べてもらいたいが，とにかくここではこのような形で図形に模様がつけられることを知ってもらいたい.

PAINT 命令は，塗りつぶすだけでなく模様もつけられることを覚えておこう.

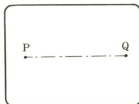
Memo　カラー模様を作るには

本節の例は 640×200 ドットのカラー画面を白黒の模様で埋めた. もしカラー模様にするには上記コーディング中の CHR$関数の中味を適当に変更すればよい.

（例）　青，赤の縞模様をつくるとき（変数Nは2とする.）

```
120 TL1$=CHR$(&HF0)+CHR$(&H0F)+CHR$(&H0)
130 TL2$=CHR$(&H0F)+CHR$(&H0F)+CHR$(&H0)
```

71 一点鎖線を描くには

二つの画面上の座標P(100, 100), Q(400, 100)を一点鎖線で結ぶにはPSET命令を用いればよい.

```
100 FOR I=100 TO 400 STEP 16
110   FOR K=0 TO 9
120     PSET (I+K,100)
130   NEXT K
140   PSET (I+12,100)
150   PSET (I+13,100)
160 NEXT I
```

この論理を用いれば, 色々な模様の線を描くことができる. しかし, 線を描くのにいちいち上記のようなコーディングをするのでは大変である. パソコンの上位機種には, これをLINE文で実行できるような機能がつけられている.

NEC PC-8801/9801では, 次のような形式でこの機能が提供されている.

LINE (X1, Y1)−(X2, Y2), C,, TILE$

このTILE$という文字変数(名前はどうでもよい)に直線のパターンを入れておくのである. たとえば上例のパターンなら次のようにする.

TILE$=CHR$ (&HFFCC)

これは次のビットパターンを画面のドットに対応させたからである.

0							7	8							15
1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	0

また長方形を模様で塗りつぶすときにも, このTILE$指定は非常に役に立つものである. LINE命令の細かい使い方ではあるが本節の内容も記憶しておくと便利である.

LINE命令は直線だけを描くのではない.

72 グラフィック画面の保存

立体図のグラフィック処理など，画面に図形を描くのに非常に時間がかかる場合がある．このようなとき，せっかく画面に描いたものを電源OFFで消してしまうのはもったいない．この節ではグラフィック画面に描かれた図形を，配列に読み込み，また配列から書き出す方法を示そう．配列に入れておけば，それをファイルに格納したりすることができて便利である．

これを実現するのに，自分のプログラムでも可能だが大変である．そこで，多くの機種では一つの命令でそれを実現できるようにしている．それが次の命令である．

```
GET@ (X1, Y1)-(X2, Y2), G%
PUT@ (X1, Y1), G%
```

GET@はグラフィック画面上の二つの座標 (X1, Y1), (X2, Y2) を対角とする長方形の内側のグラフィックパターンを配列G%に読み込む．また，PUT@はその読み込まれたデータを，グラフィック座標 (X1, Y1) を左上隅の頂点とする長方形に描く．

この二つの命令によって，グラフィックパターンが容易に配列に格納され，また利用される．描かせた図形を永久にとっておきたいときには，これをファイルにセーブすればよい．

グラフィックパターンを配列にしまっておくことは，いつでもそれを高速に画面上に再現できるということで，必然的に次の二つのメリットをも産む．

(1) 図形の平行移動が容易である

(2) 図形の運動を画面に実現できる

すなわち，(1)はGET@の (X1, Y1) とPUT@の (X1, Y1) の座標をずらせばよく，(2)は(1)の平行移動を小きざみにすればよいのである．これらの応用については他の節に譲り，ここでは一般論だけを示したにとどめておく．

計算時間を多く要したグラフィック画面はディスクに保存しておこう．

73 グラフィック座標の相対指定

下図の五角形を描くプログラムを考えてみよう．

一番容易に思いつく方法は，この図形の各頂点にグラフィック画面上の座標を与え，それを線で結ぶことである．

```
100 LINE (320,50)-(200,80)
110 LINE (200,80)-(200,150)
120 LINE (200,150)-(440,150)
130 LINE (440,150)-(440,80)
140 LINE (440,80)-(320,50)
```

上のプログラムの欠点としていえることは，もし五角形の位置を変更したいときには，プログラム全部を修正せねばならないということである．この座標の指定では，五角形を画面の中央に配置させたが，隅に移したいという修正要求が出たとき困ってしまうのである（この例のような座標の指定のし方を**絶対座標**による指定という.)．

このような欠点に対して，我々は色々な対処方法を本節までに説明してきた．

(1) WINDOW-VIEW を用いる（65節参照）

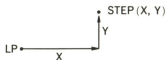
(2) 座標を変数で指定する（20 節参照）

(3) GET@，PUT@を用いる（72 節参照）

ここではこれらの方法よりもはるかに容易な方法を示そう．それが座標の**相対指定**である．実際この機能を用いて上例のプログラムを書き変えてみよう．

```
100 OX=320:OY=50
110 LINE (OX,OY) -STEP (-120,30)
120 LINE -STEP (0,100)
130 LINE -STEP (240,0)
140 LINE -STEP (0,-100)
150 LINE -STEP (-120,-30)
```

五角形の場所を変えたいときには行番号100の変数OX，OYの値だけを変更すればよいのである．

BASICはプログラムが最後に参照したグラフィック座標（LP：Last referenced point）を常にメモしている．そのメモされているLPからの相対位置でグラフィック座標を定めようとするのが，**相対座標**による座標の指定なのである．

このような指定方法を用いると，変更要求に容易に対応できるという利点とともにプログラムが非常に簡潔になるというメリットも生まれる．

LPは次のグラフィック命令を用いたときにその値を更新する．

PSET, PRESET, LINE, CIRCLE, PAINT

これらは実際に画面に点を打つ命令である．もし単にLPの値だけを変更したいときには，次の命令が用意されている．

POINT (X, Y)

また単にLPの値を知りたいときには次の関数を用いればよい．

POINT (i)　　(i=0, 1, 2, 3)

ただし，このPOINT命令（関数）についてはサポートされていないパソコンも多い．

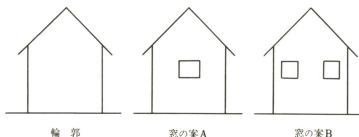
プログラムのコーディングは常に修正のしやすさを念頭に置こう．

Memo　図形の回転

本節で示したように図形の平行移動は容易であるが，回転となると話しが複雑になる．数学の行列の考え方が必要となる．これについては高校時代の数学の教科書を復習してもらいたい．

74 落書きのすすめ

パソコンでものを設計するとき，原版は残しながらもその上に色々と修正を加えたいということがよくある．たとえば，建物の輪郭を設計したが窓をその全体の中でどこに配置しようか，と考えるときなど，建物の輪郭は残して，その上で窓を色々と落書きしたくなる．

輪　郭　　　　窓の案A　　　　窓の案B

パソコンの上位機種では，多重画面を採用しているため，このような要望に対して簡単に応えることができる．以下ではNEC PC-8801/9801についてその使い方を説明しよう．

画面は640×200の白黒モードとしよう．まず次の設定をして下絵を描く．

① SCREEN 1, 0, 1, 1

下絵が描き終わったなら次のコマンドを入力する．

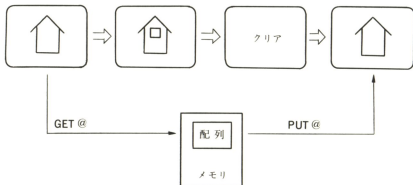
② SCREEN 1, 0, 2, 3

こうすることで，以下のプログラムでどんな絵を画面に描いても

CLS 2

を実行すると元の下絵が残り，“落書き”は消える．

NEC PC-8801/9801では，残念ながらカラーについての多重画面はできない．したがって色をつける落書きに対しては，このような方法は不可である．カラー画面で下絵を破壊しないで落書きするには次の手順をとる．

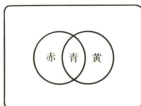
① GET@命令で画面全体を配列に読み込んでおく
② 下絵にじかに落書きをする
③ ②が終了したら下絵もろとも画面をクリアする
④ PUT@命令で画面に下絵を復元する

グラフィック画面のこのような使い方は，パソコンの応用分野を広げるのに役立つ．以上二つの技法はこの意味で大切である．

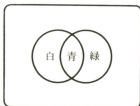
パソコンの画面をキャンバスのように用いてみよう．

Memo　二重画面と運動

画面上で図形を動かすとき，この二重画面を有効に利用すると効果が大きい．すなわち，動かした図形を画面に描いているときは前の画面をそのまま表示しておき，描き終わったら新画面を旧画面に置き換えるのである．

75 カラーコードは相対指定で

画面上に輪郭を決め，それに配色するプログラムを考える．そのプログラムを動かし画面を見てどうも配色が気にいらないとする．このとき，我々は配色を変更するわけであるが，何も工夫を凝らしていない場合，PAINT などで指定されたカラーコードを1ステップずつ変更していかねばならなくなる．たとえば，次の左図を作成するプログラムを右のように作ったとしよう．

```
100 CIRCLE (250,100),100
110 CIRCLE (370,100),100
120 PAINT (250,100),2,7
130 PAINT (300,100),1,7
140 PAINT (370,100),6,7
150 END
```

どうも配色が気に入らないと考え，次のような画面の配色を考えてみよう．そのとき修正されたプログラムを左に記しておく．

```
100 CIRCLE (250,100),100
110 CIRCLE (370,100),100
120 PAINT (250,100),7,7
130 PAINT (300,100),1,7
140 PAINT (370,100),4,7
150 END
```

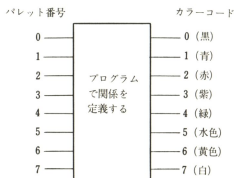
右側に書いたようなプログラムにおいて配色を修正したいときには，いちいちプログラムの中味を見て修正しようとする色コードをもつ命令を捜し，それを一つひとつ変更してゆかねばならなくなる．これでは長いプログラムの変更が大変である．

パソコンの上位機種では，このような不便さに対して色の**相対指定**を提供してくれている．NEC PC-8801/9801 では次の命令がこれに対応する．

COLOR= (I, J)

これは，プログラムにおいて，カラーコードをIとして書いたところは実際はJの物理的カラーコードで描かれることを意図している．すなわち，プログラム中は仮のIというカラーコードで書き，実際はJというコードに対応する色で塗れということを指示するものなのである．

上例のような変更に対して，このCOLOR命令を用いて，プログラミングされていれば，そのプログラム修正は容易となる．プログラム先頭で定義したこのCOLOR命令における色の対応だけを変更すればよいからである．

このCOLOR命令の（　　）の左にあるものを**パレット番号**と呼んでいる．そして次のような関係でカラーコードと対応させられる．

COLORは，コマンドレベルでも使用できる．すでに描かれている画面に，コマンドとしてこの命令を打ち込むと即座に配色が変わることになる．

コーディング時，命令のパラメータは常に相対指定となるよう心がけよう．

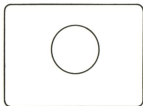
76 直線・円を消すには

画面に点を打つには PSET 命令があり，それを消すには

```
PRESET (X, Y)
```

がある．しかし，円や直線を描くには LINE 命令，CIRCLE 命令があるが，それを消すための命令はない．したがって，画面に描いた円や直線は消せないのか，というとそうではない．次のように考えればよい．

図形をバックグランドと同一色にすればその図形は消えたことになる．

いま，画面中央に白の線で円を描いたとする．

もし，この円を消そうと思うなら，同一半径，同一中心の円をバックグランドの色（いまの場合黒（カラーコード 0）とする）で描けばよいのである．

```
CIRCLE (320, 100), 100, 0
```

これで円は画面から消えた．

直線についてもまったく同様である．バックグランドの色で塗れば図形は消えるということは気がつくと当り前なのだが，それをしっかり意識していると色々と応用がきくものである．

命令を少し工夫することで多彩な活用法が見い出される．

77 グラフィック画面の一部分のクリア

画面全体を消去するには，次のような命令がある．

CLS 2 または3

しかし，画面の一部だけをクリアする命令は，それ専用にはない．たとえば左半分だけをクリアしようということは，上の命令ではできない．

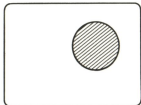
表題の内容を実行するには，次の命令を用いることを勧める．

LINE (X1, Y1)－(X2, Y2), 0, BF, 0

ここで，カラーコード（またはパレット番号）0はバックグランドの色とする．すなわち，消したい枠の内部をバッググランドと同一色で塗りつぶしてしまうのである．たとえば左半分をクリアするには次のようにする．

LINE (0, 0)－ (320, 199), 0, BF, 0

ここで画面は640×200ドットとする．

同一の原理を使えば，クリアする枠組は何も四角である必要はない．円の内側をクリアするには次のようにすればよい．

CIRCLE (400, 70), 100, 7
PAINT (400, 70), 0, 7

画面のクリアはCLS命令にこだわる必要はない．

（注） 消したいところをVIEW命令で指定し，CLS命令を実行するとその領域はクリアされる．

第6章

効率の良いプログラムの作成法

この章では計算スピードを上げる有名な計算技法を紹介する．これまでに，たびたび述べてきたようにBASICは翻訳に時間をとられ，工夫を何もしないと処理効率が非常に悪くなってしまう．本章で掲げたものはその工夫の中のごく少数のものであるが，しばしば有効に用いられるものである．ぜひ頭の中にとどめておいてもらいたい．

78 実行ステップ数は少なく

BASIC は，その言語の特性上翻訳に時間がかかる．したがって効率の良いプログラムを作成しようと思うとき，最初に思いつくのが命令数の削減である．命令の数が少なければ，その分だけコンピュータは翻訳の時間が節約できるのである．ここでは前章までに述べられてきたことも含め，この技法をまとめてみよう．

(1) BASIC 命令をたくさん覚えよう

たとえば変数 A と変数 B の値を入れ換えるのに次のようなコーディングをしたとしよう．

```
120 X=A:A=B:B=X
```

これを**三角代入法**と呼び，FORTRAN や COBOL などではしばしば用いられる論理であるが，しっかり BASIC 命令を覚えていれば次のように記述すべきである．

```
120 SWAP A, B
```

命令数が上例に比して1/3となり処理速度が向上するのである．このように，BASIC 命令を多彩に用いることで，プログラムは簡潔になり，またその分だけ処理速度が向上する．

(2) BASIC に任せられることは BASIC に任せよう

たとえば，配列 A を宣言し，その配列要素すべてを 0 に初期化しておくコーディングを次のようにしたとする．

```
140 DIM A (100)
150 FOR K=1 TO 100
160   A(K)=0
170 NEXT K
```

しかし，マニュアルをしっかり読んでいれば，次の 1 ステップでこと足りる．

```
140 DIM  A(100)
```

すなわち，BASICはDIM命令を実行すると，その宣言した配列の要素をすべて0に初期設定するのである．

この例のように，マニュアルを熟読することによってBASICに任せられることはすべてBASICに任せてしまうよう心がけるべきである．

(3) 1行ですむときは1行ですませよう

次の左右のコーディングを見てみよう．

```
10 X=A+B        10 Y=A+B+C
20 Y=X+C
```

コンパイル言語では，左右二つの実行時間に大差はない．しかしBASICでは大きな差が出てしまう．それはBASICが常に翻訳＋実行を繰り返すからである．左では二つの翻訳，右では一つの翻訳ですむ．

上例は極端な例であるが，翻訳に処理時間の多くをとられるBASICのプログラムでは，1行ですむプログラムをわざわざ2行にすることは処理効率を大幅に低下させることになることを頭に入れておいてもらいたい．

以上，コーディング上しばしば利用される注意点を三つ示したが，この他にももちろん色々な配慮を払うことによって実行ステップ数を減少させることができる．大切なことは，このことを念頭においてコーディングすることである．意識するとしないではコーディングに大きな差が出てくる．

とにかく，命令の数を少なくする努力をしよう．

79 演算回数は少なく

コンピュータも人間同様，計算をするのに時間を費やす．したがって当然計算回数は少ないにこしたことはない．ここではいくつかの有名な技法をまとめて説明しよう（すでに前章までにとり上げたものもある）．

(1) 多項式のコーディングで括弧を多用しよう

たとえば $y=x^3+3x^2+4x+2$ をコーディングするとき，次のようにコーディングするとよい．

```
200 Y=X *(X *(X+3)+4)+2
```

これを元の式のままにコーディングすると次のようになる．

```
200 Y=X * X * X+3 * X * X+4 * X+2
```

すると，上記のものに比べて * が 3 個少なくなり，その分だけ処理時間は短くなる．

(2) たびたび出る式は変数名として定義

次の例を見よう

```
100 A=SQR (1-X * X)
110 IF K=1 THEN Y=ATN (X/A)
120 IF K= 2 THEN Y=ATN(A/X)
```

これを A を用いないで記述すると

```
110 IF K=1 THEN Y=ATN (X/SQR(1-X * X))
120 IF K=2 THEN Y=ATN (SQR(1-X * X)/X)
```

一時的にしか用いない変数を上記行番号100のように使用するのは原則として控えるべきだが，（22節），この例のような場合は許されるであろうし，逆に勧められるべきである．この導入によって平方演算が 1 回，* および－演算が各々 1 回減少している．

(3) ループ内で定数となる演算は外に出そう

次の例を見よう．(これは直線を描くプログラムである．)

```
100 FOR T=-5 TO 5 STEP .1
110   X=X0+T*COS(A)
120   Y=Y0+T*SIN(A)
130   PSET (320+40*X,100+20*Y)
140 NEXT T
```

この例で COS (A), SIN (A)は，ループ内で定数である．したがって，次のように記述されるべきである．

```
100 C0=COS(A):S0=SIN(A)
110 FOR T=-5 TO 5 STEP .1
120   X=X0+T*C0:Y=Y0+T*S0
130   PSET (320+40*X,100+20*Y)
140 NEXT T
```

こうすることで，100回の SIN，COS の関数計算が節約された．

この例で分かるように，ループ内に冗長な計算があると処理速度が極端に低くなる．十分注意すべきである．

(4) 文法違反のすすめ

たとえば配列変数の添字が整数だからといっていちいち

A (INT(X))

というコーディングをする必要はない．単に A(X)で良い (注. NEC PC-8801/9801はこの技法が使えない)．このように違反してもよいような文法違反は多用して，つまらぬ演算は削減すべきである．

以上のように，ちょっとした工夫で演算回数は大幅に減少させられることを理解してもらいたい．

コンピュータも人間と同様，計算には時間がかかることを再認しよう．

80 割り算の回数は少なく

FORTRANなどのコンパイラ言語でもそうだが，BASICでは割り算よりも掛け算を多く用いるよう注意するとよい．たとえば次の数式

$$x=\frac{a}{bc}$$

をコーディングするとき，次の二つが考えられる．

```
X=A/B/C        X=A/(B * C)
```

このとき，左側は除法が2回，右側は乗法，除法が各々1回である．そして我々は右側のコーディングを勧めるのである．実際，NEC PC-9801で測定してみると，右側の方が1割ほど処理時間が短い．

特に下位機種のパソコンを利用するとき，パソコンに内蔵されているマイクロプロセッサの命令に除算がないため，除算をソフトでサポートすることになり多大な時間を乗法よりも費やすことになる．したがって，この節の技法は非常に有効なものとなる．

一般に実行速度の大小からいって，次の不等式が成立する．

＋の実行速度＞－の実行速度＞＊の実行速度＞/の実行速度

したがって，我々がプログラムをコーディングする際，できることなら除算よりも乗法を，乗法よりも減法を，減法よりも加法を用いたコーディングをすべきである．ただし，17節でも述べたように，BASIC言語の特徴からいって，この節で述べた除法以外はあまりこの技法の効果はないことを述べておこう．

除法はなるべく乗法に置き変えてコーディングしよう．

81 整数計算の商，余りは¥，MODで

多くの書物には，整数計算における商と余りの求め方の方法として次のような方法を紹介している．AをBで割った商をQ，余りをRとすると

```
100 Q=INT (A/B)
110 R=A-B * Q
```

特にFORTRANに慣れた人の著作にこの紹介が多いのはおもしろいことである．

BASICはインタプリタ言語であるという特性上，なるべく命令を簡潔に記述することが望ましく，そのためにBASICは多様な命令を我々に提供してくれている．この商，余りの計算でもBASICは次のような簡単な命令を用意してくれている．すなわち，上の命令は次のように記述できる．

```
100 Q=A¥B
110 R=A MOD B
```

この書き変えによって計算時間は半減される．

ただし，注意せねばならないことは，上記変数のA，Bが実数型であるときである．実数型変数でも，その中に入っている値が整数なら問題はないが，小数を伴うときは多くのパソコンではA，Bの小数部は切り捨てられて計算される．したがって上のプログラムで

```
A=3, B=0.4
```

のときは，0割りエラーが発生してしまう．(NEC PC-8801/9801は四捨五入される．)

¥，MOD計算はあくまで整数型の計算のためのものである．プログラム中でこの演算を用いるときには十分そのことに注意しておかねばならない．

整数計算には色々な技法・命令が多く用意されていることを知っておこう．

82 切り捨て・切り上げ・四捨五入は整数型で

正の数Xの小数点以下の切り捨て・切り上げ・四捨五入については，多くのBASICの入門書は次のような命令をのせている．

```
切り捨て　Y=INT (X)
切り上げ　Y=INT (X) : IF  X>Y  THEN  Y=Y+1
四捨五入　Y=INT (X+.5)
```

ここで，もしXの値が整数型の変数に納まる範囲，すなわち32767以下の正の数のときには，この切り捨て等の方法にもっと簡潔な技法がある．すなわち，上記命令群の中のYを整数型にしておくのである．そうすることで，次のように簡潔にコーディングされ直せる．

```
切り捨て　Y%=X
切り上げ　Y%=X : IF  X>Y%  THEN  Y%=Y%+1
四捨五入　Y%=X+.5
```

変数名の末尾に%をつけると，整数型変数となることを利用している．特に，すでに整数型宣言がなされている変数についてはもっと簡単である．すでに次の宣言がなされているとしよう．

```
DEFINT  K
```

このとき，このKを用いて上記の命令は次のようになる．

```
切り捨て　K=X
切り上げ　K=X : IF  X>K  THEN  K=K+1
四捨五入　K=X+.5
```

最初にあげたINT関数を用いたコーディングに対して，ずい分とすっきりしたものになったであろう．

このプログラミング技法は，FORTRANなどですでにおなじみのものとなっている．なぜこのようなことが可能かについては，マニュアルを読んでもらいたい．

NEC の PC-8801/9801 ではこの有名な技法を用いることはできない. K を整数型変数, X を実数型変数とすると

```
K=X
```

という等式に対して, X の小数点以下を四捨五入したものが K の中に入るように PC-8801/9801 は作られている. したがって, これらのパソコンを用いるときには次のようにコーディングする.

```
切り捨て　K=X-.5
切り上げ　K=X-.5 : IF　X>K　THEN　K=K+1
四捨五入　K=X
```

この PC-8801/9801 用のコーディングで切り捨て・切り上げについては, 何かぎこちなさを感じる. これらの機種については, プログラムの分かりやすさを損なわないために, オーソドックスな INT 関数を用いる方法の方がよいであろう.

工夫しだいで, めんどうな関数などを省くことができることを覚えておこう.

Memo　(N 桁目の) 切り捨て・切り上げ・四捨五入

N 桁目以下の切り捨て・切り上げ・四捨五入については次のようにプログラミングする.

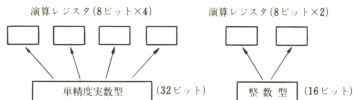
```
切り捨て　Y=INT(X/10^N)*10^N
切り上げ　Y=INT(X/10^N)*10^N : IF　X>Y　THEN　Y=Y+10^N
四捨五入　Y=INT (X/10^N+0.5)*10^N
```

この式で N=0 とおけば, 本節で述べた小数点以下の扱いになる.

小数点以下第 N 桁以降の切り捨て・切り上げ・四捨五入については, 上記の式の N に -N を代入すればよい.

83 整数演算は整数型で

結論を先に述べると，実数型演算よりも整数型演算の方が処理スピードが速い．したがって，大きくない整数（−3278 から 32767 の値）についての計算は整数型で行うべきである．

理由はこうである．多くのパソコンは 8 ビットまたは 16 ビットの演算レジスタ（実際に計算を行うところ）をもっている．したがって，実数型の場合一つの数値は 4 バイトで表現されているため，そのレジスタを 4 個ないしは 2 個連結して計算を実行せねばならなくなる．そのための準備に CPU は大きく時間をとられる．それに対して整数型は 2 バイトであり，演算レジスタは 2 個ないしは 1 個ですみ，実数型の半分ですむことになる．その分だけ CPU は実行速度を大きくする．

整数型を用いるとき，その変数名に％をつければよい．

（例）　NO％，K％，MB％

しかし見やすさ，およびコーディングのしやすさからプログラムの先頭で定義した方がよい．

（例）　DEFINT　I-N

こうすることで宣言文に宣言された頭文字をもつ変数はすべて整数形となる．この宣言のしかたであるが，我々は上例のように I から N までの頭文字をもつものを整数型とすることを勧める．これは FORTRAN の標準的な宣言と一致するが，多くのプログラマは I〜N を整数型として利用しているからであり，また多くの応用数学の書物の公式で i〜n を整数として用いているからである．

コンピュータは整数計算を得意とする．

84

(−1)^{N}の求め方

科学技術計算では，しばしば（−1）^{N}という形の計算をする．それもループ計算内部で用いられることが多いため，この計算技術をしっかり知っておくと非常に処理速度を向上させることがある．

まず素直に次のコーディングが考えられる．

```
100 X= (-1)^N                                    (1)
```

勿論 N は整数である．しかし，この記述のしかただと計算時間が N にほぼ比例する．したがって，大きな N の計算では効率が非常に悪くなってしまう．処理速度の向上を求めるとき，我々には上記の方法でなく次のコーディングを勧める．

```
100 X=1:IF N MOD 2=1 THEN X=-1                   (2)
```

こうすることで，処理速度は前記に比べて半減する（もちろん N の値にもよるが）．

この（−1）^{N}の形の計算方法には有名な方法が多くある．それをいくつか示してみよう．

```
100 X=1:IF N AND 1 THEN X=-1                     (3)
100 X= (-1)^(N AND 1)                            (4)
100 X=2*(N+1 AND 1)-.5)                          (5)
```

NEC の PC-9801 でテストしてみると（テストのしかたにもよるが），一番速いのが(2)，それに続いて(3)(5)，少し遅れて(4)，それからかなり遅れて(1)となっている．

くり返しのアルゴリズムには，必ず良い工夫があるものである．

85 簡単なメモリ節約法

パソコンは，その性格上メモリ容量が小さい．そのため，少し大きなプログラムを作ろうとすると，簡単に out of memory などというメモリオーバーのメッセージが出力されてしまう．これを避けるのは，本質的には設計段階の問題であるが，ここでは設計の基本を変えないですむメモリ節約法をまとめて示そう（この中のいくつかは前章までに紹介ずみである）．

(1) 小さい数の整数計算は整数形の変数を用いよう

整数型変数は2バイトの，単精度実数型は4バイトの，倍精度実数型は8バイトのメモリ単位で表現される．したがって，小さな整数を扱う計算は整数型で行うとメモリの節約になる．

(2) マルチ・ステートメントの勧め

1行の命令文が長くなることは，プログラムを分かりにくくする原因となるが，メモリの節約という観点からすると1行にできるだけ詰めて命令を記述した方が良いことになる．たとえば下記の左右二つのコーディングを見てみよう．

```
100  A=1          100  S=1 : X=1
110  B=1
```

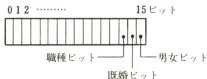
左の命令はメモリを18バイト専有するが，右のは13バイトですむ．1行にまとめることで行番号のためのメモリとテキスト制御文字のためのメモリなどが節約できたのである．

基本的には，1行に1命令を記述すべきである．しかし，1行に複数の命令（マルチステートメント）を記述することも，このようなメリットがあるため，あまりその基本のみを忠実に守る必要はないだろう．

(3) ビットに意味をもたせよう

いま，次の例を考えよう．すなわちある会社の社員のデータをコンピュータ

に記憶させることを考えてみる．色々なデータがあるが，ここでは男女の区別，既婚か未婚か，事務か技能職か，の三つを考えよう．この3個のデータはビットに意味をもたせると簡単に一変数（整数型）に収まってしまう．

整数型変数は16ビットから成り立っている．

この16ビットのうち三つのビット（上図では下位の3桁）に意味をもたせるのである．1ビットは1か0しかとれない．たとえば，男女ビットでは1が男0が女，などとあらかじめ約束しておくことで，1変数を用意しなくても十分1ビットで男女の区別ができるのである．ここで男か女かは次のようにして調べることができる．この整数型変数の名前をFとすると

男：F　AND　&H0001＝1

女：F　AND　&H0001＝0

このビット演算の値0，1がまさに男女の区別を可能にするのである．

このように，二者択一的なデータは一つの整数型変数で16種類も表現できることになる．

(4)　同時にアクセスするファイルは少なく

OPENしたファイルは，なるべくまとめて利用して，すぐにCLOSEすべきである．同時にOPENしているファイルが多いと，それに比例してBASICは作業領域を我々ユーザ領域内に確保する．その分だけメモリがとられ，我々のプログラムの入るところが小さくなってしまうのである．

以上四つの簡便なメモリ節約法を示したが，最初にも述べたように設計段階でしっかりしたメモリ予測をし，対処すべきであろう．

コンピュータは，人間のようには無限の記憶力がないことを意識しよう．

86 配列計算は帰納的に

コンピュータは帰納的な計算のしかたを得意とする．たとえば利息計算等で，必要な等比数列は次のように記述される．

```
100 N=20:DIM A(20)
110 A(1)=2:R=2
120 FOR K=1 TO N-1
130   A(K+1)=A(K)*R
140 NEXT K
```

これは初項 2，公比 2，項数 20 の等比数列の一般項 A（K）を求めるものである．もし，これを数学の公式で下記のように記述したとしよう．

```
100 N=20:DIM A(N)
110 A(1)=2:R=2
120 FOR K=1 TO N
130   A(K)=A(1)*R^(K-1)
140 NEXT K
```

上下比較すれば，明らかに上例の方が優れているだろう．すなわちむだな計算を下の例のようにしていないからである．

我々は配列を論理的に処理しようとするとき，しばしばその配列を帰納的に定義することができる．コンピュータは，まさにその形を得意とするので，そのような帰納的な定義をしっかり利用してもらいたい．

コンピュータにも計算の得手，不得手があることを知ろう!!

Memo **コンピュータの得手・不得手**

一つの論理のもとにゴシゴシ計算することは，コンピュータは得意である．しかし，色々な条件の中から最適なものを見つける，といったことはどうも苦手のようである．AI（人工知能）はまさにその苦手への挑戦であろう．

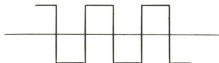
87 正負0の判定はSGN関数が便利

我々は，しばしば次のような論理をとる．

```
 :
270 IF D>0 THEN 1000
280 IF D<0 THEN 2000
290 IF D=0 THEN 3000
 :
```

このような形のとき，SGN関数を利用すると1行でことが足りる．

```
 :
270 ON SGN (D)+2 GOTO 2000, 3000, 1000
 :
```

関数SGN (X) は，X>0なら1，X=0なら0，X<0なら−1を値とするものである．したがって正か負か，といった数値の符号だけの性質を抽象化するには，大変便利な関数である．たとえば次のような台形波を描かせるにはSGNは威力を発揮する．

これは次のようにコーディングされる．

```
100 CLS 3
110 OX=0:OY=100:P=3.14159
120 LINE (0,OY)-(639,OY)
130 FOR T=0 TO 10*P STEP P/100
140   XG=10+20*T
150   S=SGN(INT(SIN(T)*20))
160   ON S+1 GOTO 180,190
170   PSET (XG,60):GOTO 200
180   LINE (XG,60)-(XG,140):GOTO 200
190   PSET (XG,140)
200 NEXT T
210 END
```

プログラミングにも定石があることを覚えよう．

88 短いプログラムが速いプログラムではない

次の例を見よう．

```
270 IF K<1 OR K>5 THEN 330
280 IF K=1 THEN R=70:C=5
290 IF K=2 THEN R=50:C=3
300 IF K=3 THEN R=30:C=7
310 IF K=4 THEN R=10:C=0
320 CIRCLE (OX,OY),R,C
330 '
```

これは IF 文が乱立し冗長なため，次のように書き変えたとしよう．

```
270 IF K<1 OR K>5 THEN 310
280 R(1)=70:R(2)=50:R(3)=30:R(4)=10
290 C(1)=5:C(2)=3:C(3)=7:C(4)=0
300 CIRCLE (OX,OY),R(K),C(K)
310 '
```

IF 文が乱立したときの回避のしかたとして有名な技法を用いた（18 節参照）．ここで，いま K=3 でこの論理に実行が移されたとする．すると上例では合計 8 個の命令（IF 命令が 5 個，R，C への代入が 2 個，グラフィック命令が 1 個）を実行するのに対し，下の例では合計 10 個（IF 命令が 1 個，代入命令が 8 個，グラフィック命令が 1 個）となり，下の方が上の方よりも 2 個実行命令が多くなる．

プログラミング技法では，明らかに下の例を勧められるが，純然と処理速度だけを考えると上の例の方が良いことになる．すなわち，簡潔なプログラムよりも冗長なプログラムの方が処理速度が速い場合があるのである．我々は美しいプログラムか，速いプログラムか，という二者択一に迫られるわけであるが，この結論はその場その場で決めてゆくべきであろう．

もう一つの例を示そう．2 節等で我々は度々プログラムはモジュール化して作成することを主張したが，このモジュール化は，一般的にいって処理速度を遅くする．たとえば，バッファ（入出力のためのメモリ）を備えたプリンタに，

計算結果を出力することを考えよう．我々は，まず先に計算し，それをファイルに保存して最後にプリンタに出力するように述べた(40節)．しかし，これについては計算しながらプリンタに打ち出した方がスピードは速い．すなわち，計算結果はすぐにプリンタのバッファに送られ，プリンタが打ち出す間にCPUは次の計算に移れるのである．したがって一回限りの計算で，かつすべてが順調にゆけば(すなわち紙詰まりなどが起こらなければ)，プログラムをモジュール化し長くすることは必要のないことになるのである．

この第2の例についても，どちらを選ぶべきかはプログラマーの判断による．しかし，経験上多くの場合，本書で主張する原則を守ったほうが結果的には"良いプログラム"になる．

原則はケースバイケースで利用してゆこう．

Memo　変数名の長さ

コンパイル言語（FORTRAN，COBOLなど）では，どんなに長い変数名をつけても，処理効率には影響を与えない．ほんの少しコンパイル時間を損するだけである．これに対してBASICでは影響を与える．まずメモリを損する．BASICは変数名をそのままの形でデータ領域に保存しておくからである．またこのことと関係するが計算時間にもわずかに影響を与えてしまう．しかし，これらの影響は通常のパソコンの処理では問題になることはなく，分かりやすい変数名をつけよ，という原則は変らない．

89 方言のすすめ

日常会話において，方言はあまり好まれてはいない．コンピュータの世界でもこの方言のために色々な問題が生じてしまう．たとえばせっかく一つの機種で開発したプログラムも，そのままでは他の機種には使えない，といった苦情がよく聞かれる．方言はプログラムの汎用性の妨げになる．

しかし，パソコンのBASICに関しては方言というものを無下に退けるわけにはゆかない．コンピュータにおいて方言が生じるにはいくつかの理由があるが，その中で各社が“便利であろう”と拡張機能として親切につけ加えてくれたものが多い．それを利用すると当然コーディングが楽になり命令数も一般的に少なくなる．BASICという言語の特性上命令数が少なくなることは，プログラムの処理効率上大いに役に立つことなのである．そこで我々は次のようにいいたい．

BASICのコーディングにおいては便利な方言を多用しよう．

たとえば次の命令を見てみよう．

```
270 CIRCLE (OX, OY), R, C1,,,,F, C2
```

これはNEC PC-8801/9801についているグラフィック命令であるが，これを“標準語”で記述すると次のようになる．

```
270 CIRCLE (OX, OY), R, C1
280 PAINT (OX, OY), C2, C1
```

明らかにコーディングが楽になり，命令数も少なくなっている．

方言が多いのは次の3つの分野である．

(1) グラフィック命令

(2) 漢字処理関係

(3) サウンド命令

特に漢字処理については，日本という国内だけの問題であり，かつ最近になって急速に発展してきたものであるから，各社まちまちの命令が各パソコンに備

えられている．逆にいえば，方言を用いなければプログラミングができないのである（(3)のサウンド命令についても事情はよく似ている）．

方言が多いのは，パソコンの学習には困ったことであり，かつ苦労して作成したソフトウェアが，他機種に利用できないという弊害があるが，当面はしかたのないことであろう．我々は各自のマニュアルをしっかり読み，これらの方言を自由に使いこなせるようにすべきであろう．

Memo ラベル

パソコンの上位機種では行番号にラベルをつけ，GOTO 命令などではこのラベルで飛べるようになっている．本節の最後の例は，このことを利用している．もしこのラベルの機能がないときにはどうするかというと，上手に注釈文を用いることをお勧めする．たとえば

```
200  IF  D<0  THEN * IMAGE
  :
240  * IMAGE
```

のところを次のようにするのである．

```
200  IF  D<0  THEN  240
  :
240  , IMAGENARY  PROCESS
```

90 ソート技法

データをある基準に従って並べ換えることをソートという．たとえば，成績や従業員番号，給料，年齢などの数値データを，数値の大きいものから小さなものへ，またはその逆に並べ換えたり，人の名前や，地名，英単語などの文字データを五十音順やアルファベット順に並べ換えたりするのがソートである．この技法は，コンピュータでデータを処理する際に頻繁に使われるもので，数ある有名なプログラミング技法の中でも特に重要な技法なのである。

データをある基準に従って並べ換える方法は一通りではない．これは数字が書いてあるたくさんのカードをでたらめに並べておいて，いろいろな人に，それらのカードを小さな数字のカードが，たとえば左側にくるように並べ換えてもらえばわかるように，人によって，実に様々な方法で並べ換えるものである．コンピュータを用いてデータを並べ換える場合にもたくさんの方法があり，色々なプログラムが考えられている．その中でも一番簡単な方法は，次の選択ソートと呼ばれているものであろう．

```
100 INPUT " ﾃﾞｰﾀｽｳ ﾊ ";N
110 DIM X(N)
120 FOR I=1 TO N:X(I)=INT(101*RND(1)):NEXT I
130 '
140 FOR J=1 TO N-1
150   FOR K=J+1 TO N
160     IF X(J)>=X(K) THEN 180
170     SWAP X(J),X(K)
180   NEXT K
190 NEXT J
200 '
210 FOR I=1 TO N:PRINT X(I):NEXT I
```

[入出力例]

```
データスウ ハ 5

98
97
61
39
2
```

この選択ソートは次の方法でデータを並べ換えている．

N 個のデータが配列 X (1), X (2), X (3), ……, X (N) に格納されているとき，まずは X (1)と X (2), X (3), …… X (N) の各々のデータを比較交換し，N 個のデータの中で，最大値を X (1)に格納する．次に X (2)と X (3), X (4), ……X (N) の各々のデータを比較交換し，X (2)から X (N) までのデータの中での最大値を X (2)に格納する．以下同様にして，X (J), X (J+1), ……X (N) の中での最大値を X (J) に格納していけば，J=N−1 のときの処理が終了した段階で，

X (1)≧X (2)≧X (3)≧……≧X (N) となる．この方法で，最大値を最小値に換えれば，X (1)≦X (2)≦X (3)≦……≦X (N) となる．

データ数が少ないときは，この選択ソートを用いて並べ換えても，コンピュータの高速処理能力によって，わずかな時間で並べ換えは終了する．しかし，データ数が大きいときには，コンピュータといえども多くの処理時間を必要とする．NEC の PC-8001mk II なるコンピュータを用いて測定した結果によると，データ数が 1000 個のときは約 1 時間 30 分もかかってしまう．これは選択ソートによると，N 個のデータを並べ換えるのに N (N−1)/2 回の比較，交換処理を必要とし，データ数が 1,000 であると，499,500 回もの比較，交換処理が行われてしまうからなのである．

このとき，次のクイックソートと呼ばれているソートプログラムで並べ換え

を行うと 1,000 個のデータを 3 分弱の処理時間で並べ換えてしまう．プログラムは複雑になってしまったが，処理時間は飛躍的に短縮化されたことになる．これは，**データの比較や交換の回数をできるだけ少なくするように工夫してあるからなのである．**

```
100 INPUT " ﾃﾞｰﾀｽｳ ﾊ ";N
110 DIM X(N),LS(N/2),RS(N/2)
120 FOR I=1 TO N:X(I)=INT(101*RND(1)):NEXT I
130 '
140 K=0:L=1:R=N
150 I=L:J=R:T=X((L+R)/2)
160 IF X(I)>T THEN I=I+1:GOTO 160
170 IF T>X(J) THEN J=J-1:GOTO 170
180 IF I<J THEN SWAP X(J),X(I):I=I+1:J=J-1:GOTO 160
190 IF I=J THEN I=I+1:J=J-1
200 IF L>=J THEN 230
210   IF I<R THEN LS(K)=L:RS(K)=J:K=K+1:L=I:GOTO 150
220   R=J:GOTO 150
230 IF I<R THEN L=I:GOTO 150
240 K=K-1
250 IF K>=0 THEN L=LS(K):R=RS(K):GOTO 150
260 '
270 FOR I=1 TO N:PRINT X(I):NEXT I
```

［入出力例］

```
ﾃﾞｰﾀｽｳ ﾊ  5

76
48
43
2
0
```

その他，いろいろなソートの方法があり，1冊の本を構成するほどである．一度はじっくり勉強してみることをおすすめしたい．

（注）　わかる超高速ソートプログラミング
　　　　高速ソート・マージ・サーチ・プログラミング法　｝（誠文堂新光社）
　　を参照されたい．

ただし，ここで注意しておきたいことは，いろいろなソートの方法があるが，それらの中で優劣はつけがたく，データの種類や用途に応じて，使いわける必要があるということである．そのためには，何種類かのソート技法を身につけておくことは有益であろう．たとえば，先のクイックソートは選択ソートよりも優れているように見えるが，データ数が少ないときは論理が簡単で，使用メモリの少ない選択ソートを使用した方が良いということになる．

Memo　マージ

2組以上の順序づけられたレコードの集団（ファイル）やデータの集団から1組の順序づけられたレコード集団（ファイル）やデータの集団をつくる操作をマージという．日本語訳では併合と呼ばれている．このマージ処理は，たとえば次のような場合に使われる．

(1)　前日までの売上げファイルや成績ファイルに，当日分のファイルをマージして，一つのファイルを作成する場合
(2)　いくつかの支店から送られてきたファイルをまとめあげて一つのファイルを作成する場合
(3)　マスターファイルに追加する場合
(4)　大量のデータをソートする場合
(5)　……

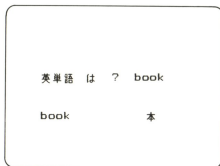
その他あげればきりがない．このマージ処理の技法もデータ処理を行う上では欠かせないものである．

91 サーチ技法

サーチとは，大量のデータの中から必要なデータを探し出すことで，日本語訳では探索と呼ばれている．たとえば，電話帳から個人名をもとにその人の電話番号を調べたり，図書目録から必要な図書を探し出したりするときのように，データバンクとしてコンピュータを使うときなどに頻繁に使われる技法である．サーチもソートと同様にいろいろな方法があり，データ処理を行う上では欠かせないものである．

一番簡単なサーチの方法は，順次探索法と呼ばれるもので，データやレコードを，それらが格納されているファイルの最初の部分から最後の部分までを一つずつシラミつぶし的な方法で探していくやり方である．たとえば，pencil, apple, orange, bird, egg, cow, zoo, book, paper, cat の順で並んでいる10個の英単語の中から，必要な単語を順次探索で探し求めて，その意味を調べるプログラムは次のようになる．

```
100 N=10:DIM X$(N,1)
110 FOR I=1 TO N
120   READ X$(I,0),X$(I,1)
130 NEXT I
140 '
150 INPUT "英単語  は   ";KE$
160 PRINT:PRINT
170 FOR I=1 TO N
180   IF X$(I,0)=KE$ THEN PRINT X$(I,0),X$(I,1)
190 NEXT I
200 END
210 '
220 DATA pencil,鉛筆
230 DATA apple,りんご
240 DATA orange,オレンジ
250 DATA bird,鳥
260 DATA egg,卵
270 DATA cow,雌牛
280 DATA zoo,動物園
290 DATA book,本
300 DATA paper,紙
310 DATA cat,ネコ
```

［入出力例］

(注) 本来，サーチ処理は，外部記憶装置に保存してあるデータに対して行うものであるが，ここでは配列 X(I) I＝1，2，3……，10 を外部記憶領域とみなして処理している．

データ数が10個ぐらいであるから，探しているデータがどこにあろうとも，この順次探索法でも，瞬時に必要なデータを探し出すが，何万，何十万とあるデータをすべて調べつくすのは，高速処理を得意とするコンピュータといえども，かなりの処理時間を必要とする．

こんなときに，効率よくデータを探し出すサーチ技法がいくつか考え出されている．そのうちの一つに，2 進探索法というものがある．

この方法は，あらかじめデータやレコードをソートしておいて，番号をつけておき（最後の番号を N 番としよう），まずは N/2 番目のデータと比較して，探しているデータが 1 番目から N/2 番目までの間にあるのか，それとも N/2 番目から N 番目までの間にあるのかを判定し，前者の間にあるときは，さらに $N/2^2$ 番目のデータと比較し，後者の間にあるときは $(N/2)+(N/2^2)$ 番目のデータと比較し，以下同様に区間を絞っていくことで，効率よく探しているデータにゆきつこうという方法である．先の英単語の例をもとにプログラミングすると次のようになる．

```
100 N=10:DIM X$(N,1)
110 FOR I=1 TO N
120   READ X$(I,0),X$(I,1)
130 NEXT I
140 '
150 INPUT "英単語 は   ";KE$
160 PRINT:PRINT
170 LS=1:RS=N
180 IF RS-LS<0 THEN PRINT "ありません":GOTO 230
190 D=INT((RS+LS)/2)
200 IF KE$<X$(D,0) THEN RS=D-1:GOTO 180
210 IF KE$>X$(D,0) THEN LS=D+1:GOTO 180
220 PRINT X$(D,0),X$(D,1)
230 END
240 '
250 DATA apple,りんご
260 DATA bird,鳥
270 DATA book,本
280 DATA cat,ネコ
290 DATA cow,雌牛
300 DATA egg,卵
310 DATA orange,オレンジ
320 DATA paper,紙
330 DATA pencil,鉛筆
340 DATA zoo,動物園
```

［入出力例］

（注） 英単語はソートされて，アルファベット順に並んでいるものとして処理している．

順次探索法だと最悪の場合は，データ数分だけ，判定処理をくり返さなけれ

ばならないが，2 進探索法だとデータ数を N としたとき最悪の場合でも $\log_2 N$ 回だけ判定処理をくり返せば，探しているデータにゆきつくことになる．N＝10,000,000（一千万）のときでも $\log_2 N$ の値は約 23.25 であり，たった 23〜4 回の判定処理ですむことになる．このくらいであれば，短時間でコンピュータはサーチ処理を完了するであろう．

また，2 進探索法の他にも，ハッシュ法と呼ばれる効率の良いサーチ技法がある．これは，辞書を我々が引くのに索引を用いることに目をつけた技法である．

（注） サーチ技法について，詳しくは高速ソート・マージ・サーチ・プログラミング法（誠文堂新光社）を参照されたい．

Memo データ・ベース

ソート・サーチ・マージは，ファイルの中のデータ構造と密接な関係がある．特にサーチ・マージについては，ファイルの中の各データがどのような関係で結びつけられているかで処理の方法が大きく変わってしまうものである．

コンピュータの世界では，ファイルの中味のデータにどのような関連をもたせ，どのようなデータ構造をとらせるかは非常に大切なことである．ある必要な情報をすぐに捜し出せ(サーチ)，またデータの追加に容易に対応できる（マージなど）ファイル・データ構成をつくるのは，容易なことではない．このようなことを研究するコンピュータの分野を，データ・ベースとよんでいる．近年情報が複雑にからみあい，かつスピーディに必要な情報を得られることがコンピュータに求められているが，そのためにはよいデータ・ベースをつくることが不可避である．

データ・ベースは，かつては大型コンピュータの分野で用いられた用語であるが，近年パソコンにも普及しはじめている．パソコンも一人前のコンピュータとして社会に認められてきたのである．

第7章

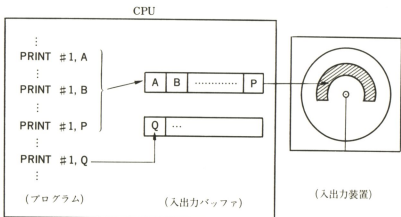
ファイル処理

処理効率を低下させる大きな要因となるものがファイル処理である．このファイル処理を如何に上手に扱うかが優れたプログラムの条件ともなるものである．ここではこのテーマについての有名な技法を紹介しよう．

92 入出力回数は少なく

数値計算処理などを除いたパソコン処理の場合，処理時間の多くはファイル処理に消費されてしまう．それはCPUの処理速度に比して周辺機器の処理が圧倒的に遅いためである．たとえばプリンタが1行印刷する間に，CPUはBASIC命令を約1,000ステップ計算してしまう．したがって，我々はプログラムを作成するとき多少CPUに負担をかけても，入出力動作をできるだけ少なくするように努めなくてはならない．ここでは，入出力動作をいかに少なくするかをシーケンシャル・ファイルとランダム・ファイルに分けて論じよう．

(1) シーケンシャル・ファイルの場合

プリンタ，テープ，ディスクにおけるシーケンシャルファイル等がこの分類に入る．このファイルは入出力命令の順に入出力を行う装置に関係するファイルである．このようなファイルは次のような手順で入出力装置にアクセスする．

この図で分かるようにBASICはCPUのメモリにあるバッファがいっぱいになって始めて入出力装置にそのバッファの内容を出力する．そして（図では二つのバッファを描いたが）次の入出力命令が出されると同じバッファ内に再び

指定されたデータを上書きしてゆく.（入力はその逆である.）

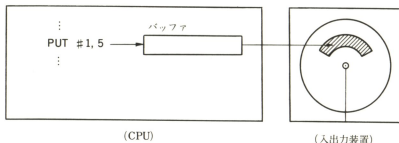
このシーケンシャルファイルの入出力方法の性質から分かるように，入出力動作を減らすには単に入出力命令を少なくするだけでは意味がない．すなわち，上図においてプログラムを次のように書き変えても無意味なのである.

PRINT #1, A, B, C, ····, Q

入出力命令は一つになって大幅に減っても入出力回数は不変である.

シーケンシャルファイルで入出力回数を減らすには，**一つのバッファに詰め込まれるデータの密度を上げること**である．プリンタなら空白をなるべく詰めるのである．テープやディスクではデータの圧縮をするのである．このようにして見かけだけでなく実効的なことをしないと，シーケンシャルファイルの入出力回数は減少しない.

(2) ランダムファイル

ランダムファイルは我々のプログラムの1回の入出力命令に1回の入出力動作が伴う.

したがって，我々のプログラム内で入出力命令の使用を少なくするとこの分だけ入出力動作は減少する.

以上のようにランダムファイルとシーケンシャルファイルとの違いをしっかり確認した上で，入出力動作の減少すなわち入出力時間の短縮をねらってもらいたい.

93 入出力データの圧縮法

92 節で説明したように，入出力に伴うデータは密度を濃くすることが有効である．それによって大幅な処理時間の短縮がなされるのである．ここではその手法をディスクファイルに限っていくつか紹介しよう．

(1) 整数データはなるべく整数型で

次の左右二つのコーディングを見てみよう．

```
PRINT  #1, A!          PRINT  #1, A%
```

小さな整数値（－32768～32767）を扱う限りでは，必ず右のコーディングのように出力変数は整数型にすべきである．これは，実数型変数 A! は 4 バイトなのに対し，整数型変数は 2 バイトですむからである．

(2) 入出力命令の特徴をつかもう

次の二つのコーディングを見てみよう．

```
PRINT  #1, A, B        PRINT  #1, A; B
```

左のようにすると，A と B とのデータ間に無意味な空白がいくつか埋められてしまう．当然右のようにすべきである．この例のように命令を熟知することでデータの圧縮が可能である．

(3) ビットに意味をもたせよう

整数型数値データは 2 バイト，実数型数値データは 4 バイト，などと変数をその形通りに使うと長さが決まってしまう．しかし，たとえばある従業員が男か女かを区別するために一つの変数を用いるのはもったいない話である．すなわち男女の区別は 0 と 1 とだけで区別できるからである．この男女の区別のように小さい整数ですむような分類等には 1 変数を割り振るのは浪費である．このような場合，整数型変数（2 バイト＝16ビット）の各ビットに意味をもたせるようにすると大幅なデータ圧縮となる．

たとえば，1から100までの番号をもった100人の男女を区別するデータを作るには次のようにできる．

```
100 DEFINT A-Z
110 N=100:DIM A(N/16)
120 FOR K=1 TO N
130   PRINT "No=";K;"  m or w ?";
140   INPUT D$
150   IF D$="m" THEN A(K¥16)=A(K¥16) OR 2^(K MOD 16)
160 NEXT K
```

こうすることで100人のデータが7変数A(0)〜A(6)に納まってしまうのである．

(4) データは整理しよう

次のコーディングを見てみよう

```
  :
1400 S=A+B
1410 PRINT #1, A; B; S
  :
```

このSは簡単にAとBから計算されたものである．したがって，このような変数Sをディスクに書き込むのはもったいないことである．データはきちんと整理し，何が本質のデータかをしっかりみぬいておかないと上のコーディングのようになってしまうのである．

以上三つの例を示したが，ちょっとした工夫でディスクに入れるデータ量は大幅に減少し，I/O時間の短縮に役立つものである．

入出力回数の削減にはデータの整理・統合が不可欠である．

94 ファイルの最後には目印を

一つのファイルがあるとき，そのファイルがどれくらい大きいのか，は次の関数で調べることができる．

LOF

これを用いるとランダムファイルの最後のレコード番号を知ることができる．また次の関数を用いるとシーケンシャルファイルの終わりを検出できる．

EOF

次の二つのプログラムは，ランダムファイル RFILE およびシーケンシャルファイル SFILE のデータを読むプログラムである．

```
100 OPEN "RFILE" AS #1
110 N=LOF(1)
120 FOR K=1 TO N
130   GET #1, K
140   :
 :    :
210 NEXT K
220 CLOSE

100 OPEN "SFILE" FOR INPUT AS #1
110 IF EOF(1) THEN 210
120   INPUT #1, A, B, C
130   :
 :    :
200 GOTO 110
210 CLOSE
 :
```

このようにシステム関数を用いることでファイルを正常に読むことは可能で

あるが，我々は次のことを敢えて要求する．すなわち我々の作成するファイルには必ず最後の目印を我々の記号でつける，という要求である．たとえばその記号を "ENDEND" とすると，次のようにするのである．

```
100 OPEN "SFILE"  FOR  OUTPUT  AS  #1
110
 :
 :      書き込み処理
 :
270 '
280 PRINT  #1,  "ENDEND"
290 CLOSE
```

これはシーケンシャルファイル "SFILE" の作成ルーチンである．このファイルを読むには次のようにする．

```
100 OPEN "SFILE"  FOR  INPUT  AS  #1
110 '
120 INPUT  #1, A$
130 IF  A$= "ENDEND"  THEN  300
 :
 :  入力処理
290 GOTO  110
300 CLOSE  #1
```

このようにすることで追加・訂正・削除等のファイル処理が非常に容易になる．また本来自分の作ったものに対しては自分で責任をもつという発想が必要であるが，EOF や LOF 関数を用いることは自分の作成したファイルを他人（システム）にまかせるという面を有しているのである．他人にまかせる部分があるとファイルの扱いをいちいちその他人に伺いを立てて実行せねばならず処理が煩らわしくなりバグ発生の原因にもなるのである．

自分のプログラムは自分で責任をとれるようにしておこう．

95 秘密保護をしっかりと

パソコンが社会に深く関与するようになると当然問題となるのが秘密の保護である．これには2種類あって，プログラム自体の保護とファイルの中のデータの保護があげられる．

(1) プログラムの機密保護

せっかく長い時間と費用をかけて作成しても，プログラムは簡単に他人にコピーされてしまう．それは音楽テープと事情が似ている．この対策については，それだけで厚い1冊の本ともなる内容であるので，ここでは次の一番簡単な方法だけを示す．それはメーカーが提供してくれたもので次のような命令を用いる．

```
SAVE "ファイル名", P
```

すなわち，プログラムをセーブするときP指定（protection）をするのである．こうすることで，このファイルの中味を単にLIST命令で見ることは不可能になる．

(2) ファイルの中のデータの秘密保護

(1)と同様に，この内容についてもここで説明を完結することは不可能である．特に近年，コンピュータ犯罪が増加しているが，多くの場合この秘密保護がしっかりなされていないことが原因であることを考えると，問題のむずかしさが分かるであろう．ここでは，一番多く使用されているパスワード（暗証番号）の方法を紹介するのにとどめよう．

パスワードでチェックするには色々な段階が考えられる．

(ア) プログラムの入口

(イ) ファイルにアクセスする前

(ウ) レコードにアクセスする前

その他レコードの内容にも機密段階を設ける，等のことも考えられるが一応こ

の三つをあげておこう．

(ア)のプログラムの入口でチェックする方法は，プログラム使用者がパソコンの前に座りプログラムをRUNさせた直後にパスワードを問う方法である．(イ)はファイルをOPENした直後に，また(ウ)はレコードを呼び出す直前にパスワードを問う方法である．

パスワードの照合のしかたとしては大きく2通りに分けられる．

(a) 合言葉形パスワード

(b) つき合わせ形パスワード

(a)はプログラムユーザ全員に同一のパスワードを教え，それを知った人間のみがチェックをパスする方法である．これには鍵をユーザ全員に渡すようなものである．これに対して(b)はユーザ個人個人が自分のパスワードを所有するのである．当然(a)よりも(b)の方がチェックが厳しくなる．

(b)の方法はランダムファイルに有効である．下のようにレコード内容を定義しておくことで容易にこの機能を実現できる．

これは社員などの個人情報が入っているレコードを想定しているが，登録番号とは社員番号のようなものとして理解してもらいたい．

このようなパスワードは，ファイルの中味が見られてしまえばすぐに見破られてしまう．そのために，これを隠す方法をとらねばならない．一つの方法は(1)で述べたソースプログラムの暗号化である．ソースプログラムが分からねばファイルのどの位置にパスワードがあるか不明である．もう一つの方法は，データ自身の暗号化である．この暗号化は過去色々と有名な方法が発見されているが，プログラム作成者が工夫を要するところである．

ファイルの中味の保護は，今後のパソコンの発展には不可欠となる．

96 ユーティリティツールの用意を

ファイル処理を伴うプログラムの開発には色々な関数，コマンド，ユーティリティプログラムを使用するものである．したがってプログラム作成者に合ったファイル処理用の道具を作っておくことを勧める．それがユーティリティツールである．これを一つのディスクの中にしまっておくのである．便利な自分の道具を一つの道具箱にしまっておくのに，それは似ている．ここでは，便利な道具のいくつかを紹介しよう．

(1) FUNCTION キーの変更ツール

システム立ち上げ時には，メーカの定義した命令がファンクションキーの中に入っている．それらの中には，我々のプログラム開発時に不要となるものも含まれている．たとえば，通常“AUTO”がメーカ定義の値として入っているが，デバッグ中には必要のないものである．そこで我々の必要なファンクションキーを定義してくれるようなプログラムをディスクに収めておくのである．たとえば次のようなものが考えられる．

```
100 '-- defkey ﾌﾟﾛｸﾞﾗﾑ --
110 KEY 2,"input"
120 KEY 3,"files"
130 KEY 6,"lprint"
140 END
```

これをディスクから呼び出し，実行させればいちいち KEY 命令を用いて入力してゆくよりもはるかに時間が早い．

(2) ユーティリティプログラムを使いやすくするツール

メーカが作成したユーティリティプログラムの名前は，メーカの開発番号などがついていて我々になじみにくいものである．したがって，次のような短いプログラムを用意しておくことを勧める．

```
100 '-- utlyty ﾌﾟﾛｸﾞﾗﾑ --
110 CLS
120 FOR K=1 TO 3:READ P$(K):NEXT K
130 DATA "2:backup.n88"
140 DATA "2:format.n88"
150 DATA "2:sysgen.n88"
160 '
170 PRINT "(1)  backup ﾌﾟﾛｸﾞﾗﾑ
180 PRINT "(2)  format ﾌﾟﾛｸﾞﾗﾑ
190 PRINT "(3)  sysgen ﾌﾟﾛｸﾞﾗﾑ
200 PRINT:INPUT "     which";ID
210 LOAD P$(ID)
220 END
```

これを実行させることで，マニュアルをいちいち引いてユーティリティプログラムの名前をいちいち入力することから開放されるのである．

(3) ファイルの中味のハードコピーをとるツール

ディスクファイル（テープファイル）は中味が見ただけではまったく分らないものである．ファイルの中味の確認等でどうしても内容の一覧リストが欲しくなるものである．それをもちろん画面に表示してもよいが，一般的にはしっかりとしたハードコピーとして残したい場合が多い．

このハードコピーツールは次の特性をもつように作成すべきである．

(ア) なるべくまとめて簡潔な形式で出力される．

(イ) ファイルの中味のすべてがみやすい形で出力される．

(イ)は当然だが，(ア)の要請は出力時間の節約から重要である．中味のコピーをとるのに何時間も費やしては大変である．

その他 WIDTH や CONSOLE 命令などをプログラミングしておき画面の初期設定等を容易にする，などが考えられるが，以上のような工夫を用いて我々はなるべく煩わしいキー操作から開放されるべきである．

煩わしいことは，パソコンにすべてまかせてしまおう．

第8章

デバッグ法

作成するプログラムが大きくなればなるほどデバッグ法が重要性を増してゆく．ここではデバッグの方法および発生に対する対処方法を示そう．

97 トレーサの活用

デバッグの助けとなる最も簡単で有効な方法は，トレーサの活用である．トレーサとはBASICがデバッグの手段として提供してくれているもので，実行を追跡（トレース）してくれる．これを使うにはコマンドとして，またはプログラム中の命令として次のものを入力する．

```
TRON
```

以後BASICはプログラムの実行を，行番号を画面に表示することで追跡してくれる．

```
10 S=0
20 FOR K=1 TO 3
30   S=S+K*K
40 NEXT K
50 PRINT S
60 END
```

（TRACERの出力）

[10] [20] [30] [40] [30] [40] [30] [40] [50] 14
[60]

行番号は［　］の内側に表示され，プログラムの出力結果と区別される．

この機能をキャンセルするときには，次のステートメントをコマンドまたはプログラム中の命令として入力する．

```
TROFF
```

このトレーサ機能を用いるとちょっとしたコーディングの論理ミスを容易に発見することが可能である．ただし，上手に制御しないと画面が瞬時に行番号で埋められ追跡がしにくくなる．また設計上の論理ミスはなかなかこの機能では捕えられない．

トレーサは使い方を工夫しないと扱いにくい．

98 テストデータの作成法

良いテストデータを作成することは，バグの少ないプログラムの作成に重要な要素となる．しかし良いテストデータは大変作るのが困難なものである．ここでは，この良いテストデータの作成について考えてみよう．

テストデータとして，しばしば用いられるのは過去のデータである．新しく手作業をパソコン処理に変えたりしたときは，手作業でやっていたデータを入力してみて，再び手作業で得られた結果と照合するのである．研究室などでも過去に結果が知られているようなデータを用意し，新しく作成したプログラムに入力し，得られた結果と既知の結果とを突き合わせてプログラムの正否を判定する．この過去のデータをテストデータとする方法は，しかし明らかに次の二つの欠点を有している．

(1) 多くの場合入力に手間どる．

(2) 新しい業務には対応できない．

特にデバッグ段階で(1)の特性は致命的となる．すなわち，テストデータの入力に手間取ることは開発期間の増大を招くからである．

この節では簡単なテストデータの作成法についてのみ限定して考える．そしてこのテストデータによるデバッグが終了してから，過去のデータがあるときはそれを入力して最終段階のデバッグとするようにしてもらいたい．

テストデータとしての要件は次のようにまとめられる．

(A) 全モジュール・全ルーチンが実行されるようなものであること

(B) 正常なデータ以外に誤りを含んだデータを用意すること

(C) 目的をしっかりもっているテストデータを多種類用意すること

(A)はテストデータの性質から当然である．(B)については，特に0割り，オーバフロー，アンダフロー，変数の型違い，等の意図的なエラーを含ませろ，ということである．(C)は(B)と関係し重複するが，デバッグ後に“この種のものに対しては既にテストが完了している”としっかりいえるようなものをテストデータとして作成せよ，ということである．

99 疑わしいところにはSTOP, PRINTを配置

プログラムの実行を追跡するトレーサ（97節参照）は，画面にすべての実行した行番号を表示するため，大きなプログラムの最初から最後まで追うことは不可能である．このことは実際画面を見ていれば分かることである．

プログラムの大きな流れをつかみ，それが正常に動作しているかを確認する一つの方法として，プログラムの節目，節目にPRINT命令を挿入することを勧める．たとえばサブルーチンの先頭に次のような命令を入れておくのである．

```
 :     :
2000  * SUBA
2010  PRINT ″SUBA  CALLED″
2020   :
 :     :
```

すると，このサブルーチンが呼ばれるたびに，メッセージが出力され，画面を見ながらプログラムの動作確認ができるのである．

デバッグが終了したら，このPRINT命令は消却すればよい．しかし，将来のバグ発生に対する対応を考えれば行の先頭にアポストロフィ（'）を挿入し保存しておいた方がよいであろう．上例では，次のようにしておくと後で見やすい．

```
2010  'PRINT ″SUBA  CALLED″ :'for DEBUG
```

すなわち，その1行にデバッグのための文であることを表示しておくのである．こうすれば，後で誰が見てもこの命令が残っていることに不思議さを感じないであろう．

デバッグのために挿入する命令として，PRINT以外にSTOP命令が有名である．STOPとは，プログラムの実行を停止する命令である．これは次のような場合に挿入すると便利である．

```
  :   :
1750  ON  K  GOTO  1770, 1810, 1880
1760  STOP : 'for DEBUG
1770  'K=1  ノ  バアイ
  :   :
```

上の例では，Kが1，2，3しか正しい動作のもとでは値としてとらないことを仮定したプログラムである．このようなとき，プログラムにバグがあり，K=0とかK>3とかの値がKに入っているとき，もし行番号1760のSTOP命令がないと大変なことになる．そのまま実行は下に抜け，妙なところでエラーが起こる．このときそのエラーの起こったところを調べても，何のバグも見当たらないことになる．そのためバグ発見に多大な時間を費やしてしまうのである．

このように，異常があっても，スルッと下に抜けてしまうようなところにはSTOP命令をちりばめておくと非常にデバッグが容易になることが分かるであろう．STOP命令が実行されプログラムが停止したときには，我々はプリント命令をダイレクトモードで用いて，いくつかの変数の中味をチェックすべきである．上の例では次のように入力するとよい．

```
PRINT  K
```

すると，Kの実際の値が分かり，どうしてKがその値をとったかの調査ができるのである．

異常検出のためにではなくとも，長いプログラムではSTOP文を入れておくとよい．たとえば，モジュールの先頭に挿入しておけば，そこに実行が移されるとプログラムは停止する．このとき，我々はモジュールに渡されるデータの正否を確かめられるのである．そして，正常なら次のコマンドを入力することで，再びプログラムの処理を続けることができる．

```
CONT
```

以上のようにプログラムをコーディングする際には**バグが発生してもすぐに見つけられるよう**にするということを心掛けるべきである．

100 バグ発生時の現場保存を

論理が複雑になると，バグが発生してもそのバグを再現するのが非常に困難になる．すなわち，色々な変数の値とちょっとしたタイミングによってバグが発生するからである．バグが発生したとき，そのプログラムを作った人がその場に居合わせればすぐに対応できるが通常はそういうことはない．したがって，バグ発生対策として**現場保存対策**を考えておかねばならない．

バグ発生時の現場保存対策として，ここでは次の方法を説明しよう．すなわち，バグが発生したらそれに対応するモジュールに実行を手渡す方法である．それを実現させる命令が次の命令である．

ON ERROR GOTO (行番号)

これは，もしBASICが検知できるエラーが生じたなら行番号に指定したルーチンに制御を渡す命令であり，次のように用いることができる．

```
RUN
1000 'PROGRAM START
1010 ON ERROR GOTO *DEBUG
1020 :
  :  :
  :  :
2700 S=S/N
  :  :
  :  (0割り発生)
4000 *DEBUG
4010 BEEP
4020 PRINT"専門家を呼んで下さい"
4030 ON ERROR GOTO 0
```

最後の命令である

ON ERROR GOTO 0

とはエラーメッセージを出力して実行を止める手続きである．

プログラム作成者（またはその理解者）が近くにいるとは限らず，その人が現れるまでパソコンを停止させるわけにはいかない場合がある．そのときのためにデバッグ処理ルーチンには次の内容を盛り込むことを勧める．

(1) 画面のハードコピー

(2) エラーコードおよびエラーの発生した行番号の印刷

(3) できるだけ多くの変数の値の印刷

(1)があればプログラムユーザがどのような使い方をしたかを知ることができ，(2)があればエラーの種類がわかる．そのエラーが発生した時点の変数の値を知ることは，デバッグに重要なことである．それが(3)である．プログラムで用いられているすべての変数をすべて出力するのは大変であるが，変数情報は多いほどよいであろう．

(1)の画面のハードコピーをとるには次の命令でよい．

```
COPY (3)
```

また(2)におけるエラーコードおよびエラー発生の行番号を知るには次の関数が便利である．

```
ERL, ERR
```

左の関数はエラー発生の行番号を，右はエラーコードを値としてとる．(3)の変数の印刷は，プログラム作成者が重要だと考える変数を一つひとつプリンタに出力するしかない．

プログラムには，必ずバグがいることを念頭においてコーディングしよう．

あとがき

BASICはパソコンのためのプログラミング言語である．言語である以上我々にはその文法だけを理解しただけでは真にBASICを分かったことにはならない．色々ないい回しや簡潔な表現方法，美しい使い方などを一つひとつ学んでゆかねばならないのである．

パソコンが社会に浸透したいま，BASICについての理解を深めることはパソコンプログラマーにとって必須の要件である．本書はそのための入門書として企画されたものだが，この目的のために多少とも役立てられれば幸いである．

しかし序でも述べたように本書を読むことで即座によいプログラミングが可能となるわけではない．最初に述べたように，BASICも言語である以上，その活用のためには各自の努力が必要なのである．その一番良い方法が，他人のプログラムの読解である．それは英文をたくさん読むと英語力がつくのと似ている．人のプログラムを多く読むことで，色々なプログラミングの新しい発想や技法を会得できるはずである．本書の内容がそのための一助となることを希望してやまない．

涌井良幸
涌井貞美

〈著者略歴〉

わくい　よしゆき
涌井　良幸

昭和25年７月13日生れ
昭和49年　東京教育大学
　　　　　理学部数学科卒業
現在　千葉県立千葉高等学校
　　　　　数学科教諭
現住所　千葉県千葉市磯辺53－５
　　　　　教職員住宅2－506

わくい　さだみ
涌井　貞美

昭和27年12月14日生れ
昭和53年　東京大学理学部
　　　　　大学院修士課程終了
現在　神奈川県立湘南高等学校
　　　　　数学科教諭
現住所　神奈川県藤沢市大庭3810
　　　　湘南ライフタウン
　　　　藤沢西部第２団地3-12-1234

PC-8801/PC-9801
BASIC ㊙ハイテク100選

NDC 548

昭和61年５月６日　発　行

定価1700円

著　者	涌　井　良　幸
	涌　井　貞　美
発行者	小　川　茂　男
発行所	株式会社 誠文堂新光社

郵便番号101　東京都千代田区神田錦町１-５-５
振替　東　京　７－6294　電話　03（292）1211
印刷　広研印刷株式会社　製本　関山製本社

検印省略　落丁・乱丁本はお取り替えいたします。

ISBN4-416-186 6-9 C2055